Operating System/2™
Programming Tools
and Information
Version 1.2

Presentation Manager Programming Reference Volume 2

64F0277

Operating System/2™
Programming Tools
and Information
Version 1.2

Presentation Manager Programming Reference Volume 2

First Edition (September 1989)

The following paragraph does not apply to the United Kingdom or any country where such provisions are inconsistent with local law:

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This publication could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time.

It is possible that this publication may contain reference to, or information about, IBM products (machines and programs), programming, or services that are not announced in your country. Such references or information must not be construed to mean that IBM intends to announce such IBM products, programming, or services in your country.

Any reference to an IBM licensed program or other IBM product in this publication is not intended to state or imply that only IBM's program or other product may be used.

Requests for technical information about IBM products should be made to your IBM Authorized Dealer or your IBM Marketing Representative.

© Copyright International Business Machines Corporation 1986, 1989. ALL RIGHTS RESERVED.

Note to US Government users - Documentation related to Restricted Rights - Use, duplication, or disclosure is subject to restrictions set forth in GSA ADP Schedule Contract with IBM Corp.

Special Notices

The following names, used in this publication, are trademarks or registered trademarks of International Business Machines Corporation in the United States and/or other countries:

IBM
Operating System/2
OS/2
Presentation Manager
Proprinter
Systems Application Architecture.

The following names, used in this publication, are trademarks or registered trademarks of other companies:

PostScript	Adobe Systems Incorporated
Windows	Microsoft Corporation.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license enquiries, in writing, to the IBM Director of Commercial Relations, IBM Corporation, Armonk, NY 10504.

About this Book

The Operating System/2 (OS/2) Version 1.2 *Presentation Manager Programming Reference* is a detailed technical reference, in two volumes, for application programmers creating programs using Presentation Manager.

For information about the Control Program, see the OS/2 Version 1.2 *Control Program Programming Reference*.

Chapter 1 contains important information. You should read it before using this book or the associated bindings references.

This reference does not give guidance on how to use the calls, nor does it contain information about how the calls are related to each other. It is intended to be used in conjunction with the OS/2 Version 1.2 *Programming Guide*.

Prerequisite Knowledge

The Programming Tools and Information library is intended for professional application developers knowledgeable in at least one programming language in which OS/2 programs can be written. The information in the Programming Tools and Information library assumes you are new to programming with OS/2 and the Presentation Manager. You should understand the OS/2 services available to users. Further information on these can be obtained from the OS/2 product library.

Related Publications

The OS/2 Version 1.2 *Programming Overview* introduces the programming concepts that you should understand before you begin developing applications to run on OS/2, and describes the set of books, tools, programming aids, and sample programs that make up the OS/2 Programming Tools and Information Library.

Bindings for the IBM C/2, IBM COBOL/2, IBM FORTRAN/2 and IBM Macro Assembler/2, languages are provided in their respective bindings references in the same order that they appear this reference. (For simplicity, C/2, COBOL/2, FORTRAN/2, and Macro Assembler/2 are referred to as "C", "COBOL", "FORTRAN", and "MASM".)

Details of publications for OS/2 Version 1.2 and related products are shown in the library diagram on page vi.

OS/2 Product

IBM Operating System/2
Standard Edition
Version 1.2

Getting Started
Using Advanced Features
Product Information

6024926 3.5″ diskette
6024930 5.25″ diskette

OS/2 Programming Tools and Information

IBM Operating System/2
Version 1.2

Installation
Programming Overview
Programming Guide
Building Programs
Dialog Tag Language Guide and Reference
Dialog Manager Guide and Reference
Dialog Manager and Dialog Tag Language Reference Summary
Programming Reference (3 books)
Bindings Reference for Presentation Manager (4 books for COBOL/2, FORTRAN/2, C/2, and Macro Assembler/2)
I/O Subsystems and Device Support (2 books)
Systems Application Architecture Common User Access: Advanced Interface Design Guide

6024929

Separate Order (no charge)

Keyboards and Code Pages

6280345

Available Separately

Command Reference

6024928

Service Coordinator's Guide

15F2214

Programming Languages

IBM Basic Compiler/2
Version 1.0 6280179

IBM Macro Assembler/2
Version 1.0 6280181

IBM Pascal Compiler/2
Version 1.0 6280183

IBM FORTRAN/2
Version 1.02 6280185

IBM COBOL/2
Version 1.0 6280207

IBM C/2
Version 1.1 6280284

Systems Application Architecture

An Overview

GC26-4341

Writing Applications: A Design Guide

SC26-4362

Common User Access: Advanced Interface Design Guide
SC26-4582

Common User Access: Basic Interface Design Guide
SC26-4583

Common Programming Interface

C Reference - Level 2

SC09-1308

COBOL Reference

SC26-4354

Dialog Reference

SC26-4356

FORTRAN Reference

SC26-4357

Procedures Language Reference
SC26-4358

Presentation Reference

SC26-4359

Organization of this Book

This book is in two volumes. The contents of each volume are as follows:

Volume One (Data Types and Function Calls)

Chapter 1, "Introduction"
This chapter contains important information about the following:

- Notation conventions
- Conventions used in function descriptions
- Error severities.

You should read it before using this book, or the associated bindings references.

Chapter 2, "Data Types"

Chapter 3, "Device Function Calls"

Chapter 4, "Graphics Function Calls"
See page 4-1 for general information about this set of calls and how they are related to each other.

Chapter 5, "Picture Function Calls"

Chapter 6, "Profile Function Calls"

Chapter 7, "Spooler Function Calls"

Chapter 8, "Advanced Video Function Calls"

Chapter 9, "Window Function Calls"
See page 9-1 for information about how these calls are related to each other.

Volume Two (Related Information)

Chapter 10, "Functions Supplied by Applications"
Procedures and hooks; see page 10-1 for general information about these functions.

Chapter 11, "Introduction to Message Processing"

Chapter 12, "Default Window Procedure Message Processing"

Chapter 13, "Button Control Window Processing"

Chapter 14, "Entry Field Control Window Processing"

Chapter 15, "Frame Control Window Processing"

Chapter 16, "List Box Control Window Processing"

Chapter 17, "Menu Control Window Processing"

Chapter 18, "Multi-Line Entry Field Control Window Processing"

Chapter 19, "Prompted Entry Field Control Window Processing"

Chapter 20, "Scroll Bar Control Window Processing"

Chapter 21, "Static Control Window Processing"

Chapter 22, "Title Bar Control Window Processing"

Chapter 23, "Clipboard Messages"

Chapter 24, "Dynamic Data Exchange Messages"

Chapter 25, "Help Manager Messages"

Chapter 26, "Resource Files"

Chapter 27, "Graphics Orders"
This chapter lists the set of graphics orders used by the Graphics Presentation Interface (GPI) calls.

Chapter 28, "Code Pages"
This chapter details the available code pages, together with an example of each one. Information about double-byte character sets is also included.

Appendix A, "Error Explanations"

Appendix B, "Standard Bit-Map Formats"

Appendix C, "The Font-File Format"

Appendix D, "Format of Interchange Files"

Appendix E, "Initialization File Information"

Appendix F, "Virtual Key Definitions."

Related Information

Chapter 10. Functions Supplied by Applications 10-1
DialogProc – Dialog Procedure 10-2
WndProc – Window Procedure 10-3
CodePageChangeHook – Code Page Change Hook 10-4
HelpHook – Help Hook 10-5
InputHook – Input Hook 10-8
JournalPlaybackHook – Journal Playback Hook 10-9
JournalRecordHook – Journal Record Hook 10-10
LoaderHook – Loader Hook 10-11
MsgCtlHook – Message Control Hook 10-13
MsgFilterHook – Message Filter Hook 10-15
RegisterUserMsg – Register User Message Hook 10-16
SendMsgHook – Send Message Hook 10-18

Chapter 11. Introduction to Message Processing 11-1
Message types 11-1
Notation Conventions 11-3

Chapter 12. Default Window Procedure Message Processing 12-1
Reserved Messages 12-1
General Window Styles 12-1
General Window Messages 12-3
Default Dialog Processing 12-50
Language Support Window Processing 12-54
Language Support Dialog Processing 12-56

Chapter 13. Button Control Window Processing 13-1
Button Control Styles 13-1
Button Control Data 13-2
Button Control Notification Messages 13-2
Button Control Window Messages 13-5

Chapter 14. Entry Field Control Window Processing 14-1
Entry Field Control Styles 14-1
Entry Field Control Data 14-2
Entry Field Control Notification Messages 14-3
Entry Field Control Window Messages 14-4

Chapter 15. Frame Control Window Processing 15-1
Standard Frame Styles and Frame Creation Flags 15-2
Frame Control Data 15-3
Frame Control Notification Messages 15-3
Frame Control Window Messages 15-6

Chapter 16. List Box Control Window Processing 16-1
List Box Control Styles 16-1
List Box Control Data 16-1
List Box Control Notification Messages 16-2
List Box Control Window Messages 16-5

Chapter 17. Menu Control Window Processing 17-1
Menu Control Styles 17-1
Menu Item Styles 17-2
Menu Item Attributes 17-2
Menu Control Notification Messages 17-3
Menu Control Window Messages 17-8

Chapter 18. Multi-Line Entry Field Control Window Processing 18-1
Multi-Line Entry Field Control Styles 18-2

Multi-Line Entry Field Control Data 18-2
Multi-line Entry Field Control Notification Messages 18-4
Multi-line Entry Field Window Messages 18-8

Chapter 19. Prompted Entry Field Control Window Processing 19-1
Combo Box Control Styles 19-1
Combo Box Control Data 19-1
Combo Box Control Notification Messages 19-2
Combo Box Control Window Messages 19-3

Chapter 20. Scroll Bar Control Window Processing 20-1
Scroll Bar Control Styles 20-1
Scroll Bar Control Data 20-1
Scroll Bar Control Notification Messages 20-2
Scroll Bar Control Window Messages 20-4

Chapter 21. Static Control Window Processing 21-1
Static Control Styles 21-1
Static Control Data 21-2
Static Control Notification Messages 21-2
Static Control Window Messages 21-3

Chapter 22. Title Bar Control Window Processing 22-1
Title Bar Control Styles 22-1
Title Bar Control Data 22-1
Title Bar Control Notification Messages 22-2
Title Bar Control Window Messages 22-3

Chapter 23. Clipboard Messages 23-1

Chapter 24. Dynamic Data Exchange Messages 24-1

Chapter 25. Help Manager Messages 25-1

Chapter 26. Resource Files 26-1
How to Read the Syntax Definitions 26-1
Definitions Used in all Resources 26-2
Resource Script File Specification 26-2
Templates, Control Data, and Presentation Parameters 26-18
Resource (.RES) File Specification 26-25

Chapter 27. Graphics Orders 27-1
Graphics Orders 27-1
Data Types 27-1
Arc at a Given Position / Arc at Current Position 27-3
Begin Area 27-3
Begin Element 27-4
Begin Image at Given Position / Begin Image at Current Position 27-4
Begin Path 27-5
Bézier Curve at Given Position / Bézier Curve at Current Position 27-6
BitBlt 27-6
Box at Given Position / Box at Current Position 27-7
Call Segment 27-8
Character String at Given Position / Character String at Current Position 27-9
Character String Extended at Given Position / Character String Extended at Current Position . 27-9
Character String Move at Given Position / Character String Move at Current Position 27-10
Close Figure 27-11
Comment 27-11
End Area 27-12
End Element 27-12
End Image 27-12
End of Symbol Definition 27-13
End Path 27-13

End Prolog 27-13
Escape 27-14
Extended Escape 27-14
Fill Path 27-15
Fillet at Given Position / Fillet at Current Position 27-15
Full Arc at Given Position / Full Arc at Current Position 27-16
Image Data 27-16
Label 27-17
Line at Given Position / Line at Current Position 27-17
Marker at Given Position / Marker at Current Position 27-18
Modify Path 27-18
No-Operation 27-19
Outline Path 27-19
Partial Arc at Given Position / Partial Arc at Current Position 27-19
Pop 27-20
Relative Line at Given Position / Relative Line at Current Position 27-20
Set Arc Parameter / Push and Set Arc Parameter 27-21
Set Background Color / Push and Set Background Color 27-22
Set Background Indexed Color / Push and Set Background Indexed Color 27-22
Set Background Mix / Push and Set Background Mix 27-23
Set Character Angle / Push and Set Character Angle 27-24
Set Character Cell / Push and Set Character Cell 27-25
Set Character Direction / Push and Set Character Direction 27-26
Set Character Precision / Push and Set Character Precision 27-26
Set Character Set / Push and Set Character Set 27-27
Set Character Shear / Push and Set Character Shear 27-27
Set Clip Path 27-28
Set Color / Push and Set Color 27-29
Set Current Position / Push and Set Current Position 27-29
Set Extended Color / Push and Set Extended Color 27-30
Set Fractional Line Width / Push and Set Fractional Line Width 27-30
Set Indexed Color / Push and Set Indexed Color 27-31
Set Individual Attribute / Push and Set Individual Attribute 27-32
Set Line End / Push and Set Line End 27-33
Set Line Join / Push and Set Line Join 27-33
Set Line Type / Push and Set Line Type 27-34
Set Line Width / Push and Set Line Width 27-35
Set Marker Cell / Push and Set Marker Cell 27-35
Set Marker Precision / Push and Set Marker Precision 27-36
Set Marker Set / Push and Set Marker Set 27-36
Set Marker Symbol / Push and Set Marker Symbol 27-37
Set Mix / Push and Set Mix 27-38
Set Model Transform / Push and Set Model Transform 27-39
Set Pattern Reference Point / Push and Set Pattern Reference Point 27-40
Set Pattern Set / Push and Set Pattern Set 27-40
Set Pattern Symbol / Push and Set Pattern Symbol 27-41
Set Pick Identifier / Push and Set Pick Identifier 27-42
Set Segment Boundary 27-42
Set Segment Characteristics 27-43
Set Stroke Line Width / Push and Set Stroke Line Width 27-44
Set Viewing Transform 27-44
Set Viewing Window / Push and Set Viewing Window 27-45
Sharp Fillet at Given Position / Sharp Fillet at Current Position 27-46

Chapter 28. Code Pages 28-1
Full Screen VIO Applications 28-1
Windowed PM Applications 28-2
OS/2 Version 1.2 Code Page Options for VIO and PM Applications 28-3
OS/2 Version 1.2 Font Support For Multiple Code Pages 28-5
ASCII Code Pages 28-7
EBCDIC Code Pages 28-12
DBCS Support 28-18

Chapter 10. Functions Supplied by Applications

This chapter describes dialog procedures, window procedures, and hooks. It shows the input parameters and returns that IBM Operating System/2 expects an application to use in application procedures, that can be called by IBM Operating System/2, in response to certain events.

An API is an interface where the application calls the system, whereas procedures and hooks are interfaces where the system calls the application.

The names and parameter list of APIs are contained in header files that are incorporated into the application when it is compiled. Their addresses are contained in .LIB files that are incorporated at link time.

The names of procedures and hooks are defined by the application, and their parameter lists are defined by the system. Function prototypes for these procedures and hooks are in PMWIN.H. The prototypes have sample names that can be changed by the programmer before they are inserted into the application source code.

The application passes the address of these procedures and hooks in the following ways:

Dialog procedures During the WinLoadDlg or WinDlgBox APIs

Window procedures During the WinRegisterClass or WinSubclassWindow APIs

Hooks During the WinSetHook API.

Procedures and Hooks are shown in the following table:

C/MASM name	COBOL/FORTRAN name
Procedures	
DialogProc	not used
WndProc	not used
Hooks	
CodePageChangeHook	not used
HelpHook	not used
InputHook	not used
JournalPlaybackHook	not used
JournalRecordHook	not used
LoaderHook	not used
MsgCtlHook	not used
MsgFilterHook	not used
RegisterUserMsg	not used
SendMsgHook	not used

DialogProc — Dialog Procedure

SAA

This is a window procedure that automatically subclasses each instance of a dialog box.

DialogProc (hwnd, msg, Param1, Param2, *Reply***)**

Parameters

hwnd *(HWND)* — input
Handle of the window to which the message applies.

msg *(USHORT)* — input
Message identity.

Param1 *(MPARAM)* — input
Message parameter 1.

Param2 *(MPARAM)* — input
Message parameter 2.

Reply *(MRESULT)* — return
Message-return data.

Remarks

This procedure is the same as any other window procedure, except that it can receive predefined window messages specific to dialog box windows.

Programming Note: It does **not** receive the WM_CREATE message, but the same information is carried by the WM_INITDLG message, that is generated during the creation of a dialog-box window.

hwnd is always the window handle of the dialog-box window.

The dialog procedure typically processes only some of the messages passed to it. Any messages that it does not process must be passed to WinDefDlgProc (not WinDefWindowProc), because this performs the standard dialog-box processing for those messages.

WndProc — Window Procedure

This defines the window procedure provided by an application.

WndProc (hwnd, msg, Param1, Param2, *Reply*)

Parameters

hwnd *(HWND)* — input
Window handle.

msg *(USHORT)* — input
Message identity.

Param1 *(MPARAM)* — input
Message parameter 1.

Param2 *(MPARAM)* — input
Message parameter 2.

Reply *(MRESULT)* — return
Message-return data.

Remarks

This procedure is associated with a window by the **WndProc** of the WinRegisterClass function.

The window procedure typically processes only some of the messages passed to it. Those messages it does not process must be passed on to the WinDefWindowProc function, which performs the standard window processing for those messages.

CodePageChangeHook – Code Page Change Hook

This hook notifies that a message queue code page has been changed.

```
CodePageChangeHook (hmq, OldCodePage, NewCodePage, return)
```

Parameters

hmq *(HMQ)* – input
Message-queue handle.

The handle of the message queue that is changing its code page.

OldCodePage *(USHORT)* – input
Previous code page.

NewCodePage *(USHORT)* – input
New code page.

return *(VOID)* – return
Return value.

The return value from this hook is void.

Remarks

This hook is sent to all hooks chained under HK_CODEPAGECHANGE, regardless of the return value.

The new code page is set before this hook is called.

HelpHook — Help Hook

This hook processes help requests.

```
HelpHook (hab, Mode, Topic, SubTopic, Position, f)
```

Parameters

hab *(HAB)* — input
Anchor-block handle.

Mode *(SHORT)* — input
Help mode.

This has one of the following values, indicating the mode from which help has been requested:

HFM_MENU	Menu mode
HFM_MB	Message-box mode
HFM_WINDOW	Standard (standard window)
HFM_APPLICATION	Application mode.

Topic *(SHORT)* — input
Topic identifier.

- In menu mode this is a pull-down window identity
- In message-box mode this is the message-box identity
- In standard mode this is a window identity.

SubTopic *(SHORT)* — input
Subtopic identifier.

- In menu mode this is a command identity
- In message-box mode this is a control identity
- In standard mode this is the identity of the window with the focus (**–1** if none).

Position *(RECT)* — input
Rectangle.

This indicates the screen area (in screen coordinates) from where the help was requested. It is provided to enable the help library to avoid covering that area.

- In menu mode it is the bounding rectangle of the selected item,or of the top level menu if value of the **SubTopic** parameter is **–1**.
- In message-box mode it is the bounding rectangle of the button.
- In standard mode it is the bounding rectangle of the window with the focus, or of the window sent the message if the value of the **SubTopic** parameter is **–1**.

Note: The data type *WRECT* can also be used, if supported by the language.

f *(BOOL)* — return
Indicator as to whether next hook in the chain is called.

The message is always passed to the application.

TRUE	The next hook in the chain is not called
FALSE	The next hook in the chain is called.

HelpHook –
Help Hook

Remarks

This hook can be called directly by an application or in the default-processing associated with windows, menus, and message boxes.

Help-processing is done in two stages. The first stage is the creation of the WM_HELP message. This is done:

- From a WM_CHAR message by ACCELERATOR table translation, when the HELP accelerator option is specified.
- From an action-bar selection, when the MIS_HELP style is specified on the action-bar button.
- From a dialog-box pushbutton, when the BS_HELP style is specified on the pushbutton.
- From a message box, when the MB_HELP style is specified on the message box.

The WM_HELP message is sent to the active window, but will be seen by a modal loop if one is active.

The second stage of processing of help is the processing of the WM_HELP message.

The frame window procedure sees the WM_HELP message because the frame is usually the active window. It processes the WM_HELP message as follows:

- If the window with the focus is the FID_CLIENT frame control, it passes WM_HELP to the FID_CLIENT window.
- If the parent of the window with the focus is the FID_CLIENT frame control, it calls the help hook, specifying:

 Mode = HFM_WINDOW
 Topic = frame-window id
 SubTopic = focus-window id.

- If the parent of the focus window is not the FID_CLIENT frame control (for example, it may be the frame itself, or a second-level dialog control), it calls the hook, specifying:

 Mode = HFM_WINDOW
 Topic = focus-window parent id
 SubTopic = focus-window id.

The message box window procedure sees the WM_HELP message, because it subclasses the frame window. It processes the WM_HELP message by calling the help hook, specifying:

Mode = HFM_MESSAGE
Topic = message id
SubTopic = control id.

The menu window procedure sees the WM_HELP message because it runs a modal loop. It processes the WM_HELP message by calling the help hook, specifying:

Mode = HFM_MENU
Topic = menu id of pull-down
SubTopic = menu id of item.

The WinDefWindowProc function sees the WM_HELP message for a FID_CLIENT window if the client does not handle it itself. It calls the help hook, specifying:

Mode = HFM_WINDOW
Topic = active-window id
SubTopic = focus-window id.

An application sees the WM_HELP message in its dialog procedure. The application can ignore the WM_HELP message, in which case the frame-window procedure action occurs (as described above) or it can simulate a call to the help hook itself, using:

Mode = HFM_APPLICATION
Topic = any value
SubTopic = any value.

The input focus is never given to any of the standard frame controls, so help for these cannot be obtained.

InputHook – Input Hook

This hook filters messages from the input queue.

InputHook (hab, Qmsg, Options, *Processed*)

Parameters

hab *(HAB)* – input
Anchor-block handle.

Qmsg *(QMSG)* – input
A QMSG data structure.

Options *(BIT16)* – input
Message removal options:

PM_REMOVE	Message is being removed from queue
PM_NOREMOVE	Message is not being removed from queue.

Processed *(BOOL)* – return
Processed indicator:

TRUE	The message is not passed on to the next hook in the chain or to the application
FALSE	The message is passed on to the next hook in the chain or to the application.

Remarks

This hook is called when messages are removed from an application queue, before being returned by WinGetMsg or WinPeekMsg. It is called from within these functions just before resuming the application with the message that is returned. There are no restrictions on calls that may be made at this time.

JournalPlaybackHook — Journal Playback Hook

This hook plays back recorded messages.

JournalPlaybackHook (hab, qmsg, Skip, *Time*)

Parameters

hab *(HAB)* — input
Anchor-block handle.

qmsg *(QMSG)* — input
Data structure where the message to be played back is returned.

When this hook is called, the **time** field of the *QMSG* structure is initialized to the current time. This can be used to determine whether the next message is ready or not. This value must be used for any delta calculations performed by the hook procedure, rather than the result of WinGetCurrentTime.

Skip *(BOOL)* — input
Indicator as to whether the next message should be played back:

TRUE The journal playback hook skips to the next message. The **qmsg** parameter is NULL in this case. The next hook in the chain is not called.

FALSE The journal playback hook returns the next available message. The same message is returned each time, until it is skipped with a call where this parameter is TRUE.

Time *(LONG)* — return
Waiting time.

The time to wait (in milliseconds) before processing the current message.

Remarks

This hook is called whenever a message is required to be played back.

JournalRecordHook — Journal Record Hook

This hook records user-input messages.

```
JournalRecordHook (hab, qmsg, Success)
```

Parameters

hab *(HAB)* — input
Anchor-block handle.

qmsg *(QMSG)* — input
Data structure that contains the message to be recorded.

The **hwnd** field of the *QMSG* structure is also set when the hook is called.

Success *(BOOL)* — return
The return value from this hook is ignored.

Remarks

This hook is called **after** raw input is translated to WM_CHAR or WM_BUTTON1DBLCLK messages.

The next hook in the chain is always called, and the message is always passed to the application.

JournalPlaybackHook does not receive any input played back by this hook. This prevents feedback situations where input is played back a number of times.

LoaderHook — Loader Hook

This hook allows the library and procedure loading and deleting calls to be intercepted.

LoaderHook (hab, Context, libname, Libhandle, procname, wndproc, Success, *Processed*)

Parameters

hab *(HAB)* — input
Anchor-block handle.

Context *(SHORT)* — input
Origin of call to hook.

LHK_DELETELIB	WinDeleteLibrary
LHK_DELETEPROC	WinDeleteProcedure
LHK_LOADLIB	WinLoadLibrary
LHK_LOADPROC	WinLoadProcedure.

libname *(STRL)* — input
Library name.

This is the same as the library name in the **Libname** parameter of the WinLoadLibrary call.

Libhandle *(HLIB)* — input/output
Library handle.

This is the same as the library handle in the **Libhandle** parameter of the WinLoadProcedure call or the **Libhandle** parameter of the WinDeleteLibrary call or

If the **Context** parameter is set to LHK_LOADLIB, then this hook must set the value of this parameter to the handle of the loaded library or to NULL if the load fails.

procname *(STRL)* — input
Procedure name.

This is the same as the procedure name in the **Procname** parameter of the WinLoadProcedure call.

wndproc *(PROC)* — input
Window procedure identifier.

This is the same as the library name in the **wndproc** parameter of the WinDeleteProcedure call.

If the **Context** parameter is set to LHK_LOADPROC, then this hook must set the value of this parameter to the handle of the loaded procedure or to NULL if the load fails.

Success *(BOOL)* — input/output
Success indicator:

TRUE	Library or procedure loaded or deleted successfully
FALSE	Library or procedure not loaded or deleted successfully.

Processed *(BOOL)* — return
Processing indicator:

TRUE	Do not call next hook in chain
FALSE	Call next hook in chain.

LoaderHook – Loader Hook

Remarks

If the hook attempts a load or deletion which is unsuccessful, then the WinSetErrorInfo must be used to establish the relevant error information.

MsgCtlHook – Message Control Hook

This hook allows the call which determine the flow of messages to be intercepted.

MsgCtlHook (hab, Context, hwnd, ClassName, MsgClass, Control, Success, *Processed***)**

Parameters

hab *(HAB)* – input
Anchor-block handle.

Context *(SHORT)* – input
Origin of call to hook.

MCHK_CLASSMSGINTEREST	WinSetClassMsgInterest
MCHK_MSGINTEREST	WinSetMsgInterest
MCHK_MSGMODE	WinSetMsgMode
MCHK_SYNCHRONISATION	WinSetSynchroMode.

hwnd *(HWND)* – input
Window handle.

This is the same as the window handle in the **hwnd** parameter of the WinSetMsgInterest call.

ClassName *(STRL)* – input
Window class name.

This is the same as the window class name in the **ClassName** parameter of the WinSetClassMsgInterest call.

MsgClass *(USHORT)* – input
Message class.

This is the same as the message class in the **MsgClass** parameter of the WinSetMsgInterest and the WinSetClassMsgInterest calls.

Control *(SHORT)* – input
Control setting.

The setting varies with the value of the **Context** parameter.

For MCHK_CLASSMSGINTEREST, it can be SMI_INTEREST, or SMI_NOINTEREST, or SMI_AUTODISPATCH.

For MCHK_MSGINTEREST, it can be SMI_INTEREST, or SMI_NOINTEREST, or SMI_RESET, or SMI_AUTODISPATCH.

For MCHK_MSGMODE, it can be SMD_DELAYED or SMD_IMMEDIATE.

For MCHK_SYNCHRONISATION, it can be SSM_SYNCHRONOUS, or SSM_ASYNCHRONOUS, or SSM_MIXED.

Success *(BOOL)* – input/output
Success indicator:

TRUE	Mode or interest successfully set
FALSE	Mode or interest not successfully set.

Processed *(BOOL)* – return
Processing indicator:

TRUE	Do not call next hook in chain
FALSE	Call next hook in chain.

MsgCtlHook – Message Control Hook

Remarks

If the hook is unable to alter the message control state, then the WinSetErrorInfo must be used to establish the relevant error information.

MsgFilterHook — Message Filter Hook

This hook filters messages from inside a mode loop.

MsgFilterHook (hab, Qmsg, Context, *Processed*)

Parameters

hab *(HAB)* — input
Anchor-block handle.

Qmsg *(QMSG)* — input
A queue message data structure.

Context *(USHORT)* — input
Context in which the hook has been called:

MSGF_DIALOGBOX	Dialog-box mode loop.
MSGF_MESSAGEBOX	Message-box mode loop.
MSGF_TRACK	Window-movement and size tracking. When this hook is used the *TRACKINFO* structure specified the **TrackInfo** parameter of the WinTrackRect call is updated to give the current state before the hook is called. Only the **track** and the **options** parameters are updated.
MSGF_MENU	Menu tracking.
MSGF_SCROLLBAR	Scroll-bar tracking.
MSGF_NEXTWINDOW	Window-enumeration mode loop.

Processed *(BOOL)* — return
Processed indicator:

TRUE	The message is not passed on to the next hook in the chain or to the application
FALSE	The message is passed on to the next hook in the chain or to the application.

Remarks

This hook is called inside any of the system-mode loops, for instance, during size-tracking or move-tracking, or while a dialog box or menu is displayed.

The WM_QUIT message is passed to this hook, if it occurs during a mode loop.

RegisterUserMsg – Register User Message Hook

This hook allows user messages and user data types to be registered.

RegisterUserMsg **(hab, Context, Msgid, Type1, Dir1, Type2, Dir2, Typer, Count, Types, Success,** *Processed*)

Parameters

hab *(HAB)* – input
Anchor-block handle.

Context *(SHORT)* – input
Origin of call to hook.

RUMHK_DATATYPE	Register User Data type
RUMHK_MSG	Register User Message.

Msgid *(USHORT)* – input
Message identifier.

If the origin of the call is 'Register User Data Type', this parameter is not set.

Type1 *(SHORT)* – input
Data type of message-parameter 1.

If the origin of the call is 'Register User Data Type', this parameter contains the data type code to be registered.

Dir1 *(SHORT)* – input
Direction of message-parameter 1.

If the origin of the call is 'Register User Data Type', this parameter is not set.

Type2 *(SHORT)* – input
Data type of message-parameter 2.

If the origin of the call is 'Register User Data type', this parameter is not set.

Dir2 *(SHORT)* – input
Direction of message-parameter 2.

If the origin of the call is 'Register User Data Type', this parameter is not set.

Typer *(SHORT)* – input
Data type of message reply.

If the origin of the call is 'Register User Data Type', this parameter is not set.

Count *(SHORT)* – input
Number of elements.

If the origin of the call is 'Register User Message', this parameter is not set.

Types *(SHORT****Count)** – input
Data types of structure components.

If the origin of the call is 'Register User Message', this parameter is not set.

Success *(BOOL)* – input/output
Success indicator:

TRUE	Successful completion
FALSE	Error occurred.

Processed *(BOOL)* – return
Processing indicator:

TRUE Do not call next hook in chain
FALSE Call next hook in chain.

SendMsgHook – Send Message Hook

This hook filters messages sent by the WinSendMsg function.

SendMsgHook (hab, smh, InterTask, *return***)**

Parameters

hab *(HAB)* – input
Anchor-block handle.

smh *(SMHSTRUCT)* – input
Send message hook structure.

This parameter is a structure that contains the parameters to the WinSendMsg function.

InterTask *(BOOL)* – input
Intertask indicator:

TRUE The message is sent between tasks (intertask)
FALSE The message is sent within a task (intratask).

return *(VOID)* – return
Return value.

The return value from this hook is void.

Remarks

This hook may be called whenever a window procedure is called via the WinSendMsg function.

It is called in the context of the sender, whereby if the sender has a queue hook installed it is called, but if the receiver has a queue hook installed it is not called.

The next hook in the chain is always called.

Chapter 11. Introduction to Message Processing

Messages are processed by window and dialog procedures.

Every window has a window procedure. Windows can also be combined into standard windows or dialog boxes. These are special cases of groups of windows that also have their own procedures. A window or dialog procedure must be capable of processing any message. This can be achieved by delegating some message types to the default window, or dialog, procedures by use of the WinDefWindowProc and WinDefDlgProc calls respectively.

All messages have the following form:

QMSG Message structure.

hwnd *(HWND)*
Window handle.

msgid *(USHORT)*
Message identity.

param1 *(MPARAM)*
Parameter 1.

param2 *(MPARAM)*
Parameter 2.

time *(ULONG)*
Message time.

point *(POINTL)*
Pointer position when message was generated.

Message types

There are two types of window procedure message processing.

- Default window and dialog procedure message processing
- Control window message processing.

These types are described below along with the notation conventions used in the message descriptions. The messages are described in the following chapters.

Default Window Procedure Message Processing

These window procedures provide default processing for application window procedures:

- Default window and dialog procedure
- Language support window and dialog procedures, which are used if the application specifies a "NULL" window procedure
- Default AVIO window procedure.

These messages are described in Chapter 12, "Default Window Procedure Message Processing." The system-provided window procedures take no action on messages that are not defined in this chapter, and return NULL.

Control Window Message Processing

Controls are predefined classes of child windows that any application can use for input and output. These control classes are predefined:

WC_BUTTON Consist of buttons and boxes that the operator can select by clicking the pointing device or using the keyboard. These messages are described in Chapter 13, "Button Control Window Processing."

WC_COMBOBOX Consists of an entry field control and a list box control merged into a single control. The list, which is usually limited in size, is displayed below the entry field and offset one dialog box unit to its right. These messages are described in Chapter 19, "Prompted Entry Field Control Window Processing."

WC_ENTRYFIELD Consist of a single line of text that the operator can edit. These messages are described in Chapter 14, "Entry Field Control Window Processing."

WC_FRAME Composite window. These messages are described in Chapter 15, "Frame Control Window Processing."

WC_LISTBOX Present a list of text items from which the operator can make selections. These messages are described in Chapter 16, "List Box Control Window Processing."

WC_MENU Present a list of items, which may be text displayed horizontally as action bars or vertically as pull-down menus. Menus are usually used to provide a command interface to applications. These messages are described in Chapter 17, "Menu Control Window Processing."

WC_MLE An multi-line entry field control is a rectangular window that displays multiple lines of text that the operator can edit. When it has the focus, the cursor marks the current **insertion** or **replacement** point. These messages are described in Chapter 18, "Multi-Line Entry Field Control Window Processing."

WC_SCROLLBAR Consist of window scroll bars that allow the operator to make a request to scroll the contents of an associated window. These messages are described in Chapter 20, "Scroll Bar Control Window Processing."

WC_STATIC Simple display items that do not respond to keyboard or pointing device events. These messages are described in Chapter 21, "Static Control Window Processing."

WC_TITLEBAR Displays the window title or caption and allows the operator to move its owner. These messages are described in Chapter 22, "Title Bar Control Window Processing."

Owner-Notification Messages: Controls are useful because they notify their owners when significant events take place. A control notifies its owner by sending a WM_CONTROL message or by posting a WM_COMMAND or WM_HELP message.

- WM_CONTROL
- WM_COMMAND

 Param2 contains information that indicates the source of the WM_COMMAND message:

 CMDSRC_PUSHBUTTON Posted by a pushbutton control
 CMDSRC_MENU Posted by a menu control
 CMDSRC_ACCELERATOR Posted by WinTranslateAccel
 CMDSRC_OTHER Other source.

- WM_HELP

 Param2 contains information that indicates the source of the WM_HELP message:

 CMDSRC_PUSHBUTTON Posted by a pushbutton control
 CMDSRC_MENU Posted by a menu control
 CMDSRC_ACCELERATOR Posted by WinTranslateAccel
 CMDSRC_OTHER Other source.

Notation Conventions

Each message description contains:

Name The message name; a 2-byte identity unique to a message. Messages generated by the system have an identity below the constant WM_USER; see "Reserved Messages."

Applications generating their own messages must use a value higher than WM_USER.

For all messages, the first two or three characters of the name indicate the type of window that is related to the message; for example:

LM List box control
SBM Scroll bar control.

Cause The principal reason that caused the generation of the message.

Parameters Input and output parameters pertinent to the message.

There are always two parameters (**param1** and **param2**) and one **return** value. Any or all of the parameters can be NULL.

Remarks An explanation of the relationship between the parameters in the context of the message and an indication of the expected processing of the message.

Default A definition of how the default window procedures (provided by the system) process the message.

Programming Note: A message is not equivalent to a call of the same name.

Chapter 12. Default Window Procedure Message Processing

Reserved Messages

These message ranges are reserved:

WM_USER All messages below this value are reserved for system use. Private messages should have an identifier with a value of WM_USER or higher.

General Window Styles

The *window* is the mechanism by which the application communicates with the operator. Each window can have a window "style" that controls the appearance and behavior of the window. There are also "class" styles that apply to all the windows of a particular class (class being FRAME, BUTTON, and so on).

These styles are listed below.

Window Class Styles

Style	Meaning
CS_SIZEREDRAW	This style determines whether a window should be redrawn when sized. This style should be used for a window whose contents are sensitive to the size of the window. For example, the data in some windows can be scaled up or down to fit the size of the Client Area. In other windows, the data remains the same size whatever the size of the window; it is merely clipped if the window is made smaller. The CS_SIZEREDRAW style should be used in the first instance but not in the second. For more information, see WM_CALCVALIDRECTS.
CS_SYNCPAINT	Window is synchronously repainted. This style causes WS_SYNCPAINT to be set for all windows of this class.
CS_MOVENOTIFY	This class style should be used by a child window if it wants to be notified with a WM_MOVE message when its parent is moved. For more detail, see the WM_MOVE message description.
CS_CLIPCHILDREN	Causes a window of style WS_CLIPCHILDREN to be created, regardless of whether this style bit is specified on the create window call.
CS_CLIPSIBLINGS	Causes a window of style WS_CLIPSIBLINGS to be created, regardless of whether this style bit is specified on the create window call.
CS_PARENTCLIP	Causes a window of style WS_PARENTCLIP to be created, regardless of whether this style bit is specified on the create window call.
CS_SAVEBITS	Causes a window of style WS_SAVEBITS to be created, regardless of whether this style bit is specified on the create window call.
CS_PUBLIC	Causes a public window class to be registered. It is an error if this parameter is specified on any process other than the shell process.

CS_HITTEST	If set, causes a WM_HITTEST message to be sent to the window, before sending any pointing device message. If not set, no WM_HITTEST message is sent, and it is assumed that the window returns HT_NORMAL if the window is not disabled, and HT_ERROR if the window is disabled. Top-level frame windows do not have CS_HITTEST set.
CS_FRAME	If set, all windows of this class are expected to behave as frame windows.

Window Styles

Style	Meaning
WS_SYNCPAINT	Window is synchronously repainted. This style is set for windows that have Class Style CS_SYNCPAINT. Applications can then turn this style on and off to vary the window processing.
System-Provided Window Styles	
WS_CLIPCHILDREN	This specifies that the area occupied by a window's children is to be excluded when drawing in that window. Normally, it is included.
WS_CLIPSIBLINGS	This specifies that the area occupied by a window's siblings is to be excluded when drawing in that window. Normally, it is included.
WS_DISABLED	This specifies that the window is disabled. The default is enabled.
WS_MAXIMIZED	This specifies that the frame window is to be created maximized. When a window is moved or sized in the normal way at least one border should remain on the screen. When a window is maximized and the maximum size is as large as the screen all borders should be positioned just outside the screen.
WS_MINIMIZED	This specifies that the frame window is to be created minimized.
WS_PARENTCLIP	This controls how a window is clipped when a drawing action takes place into the window. Generally, a WS_PARENTCLIP window should not draw outside its window rectangle.
WS_SAVEBITS	This specifies that the screen image of the area under a window of this style be saved when the window is made visible.
WS_VISIBLE	This specifies that the window is visible. The default is invisible. **Note:** A window can still be visible, in this sense, even if it cannot be seen because it is covered by other windows.
Styles for Windows in Dialogs	
WS_GROUP	This identifies the dialog items that make up a 'group'. This style should be specified on the first window of any group. Subsequent windows of the group must not have this style. The windows of the group must be adjacent siblings. This can be done by listing the windows consecutively in templates (see "Dialog Template" on page 26-18) or by inserting each new window in the group behind the previous one (WinCreateWindow).
WS_TABSTOP	This identifies a dialog item as one to which the operator can TAB.

General Window Messages

PL_ALTERED

This message is broadcast when the PrfReset call is issued.

Parameters

param1

user *(HINI)*
Handle of the new user profile.

param2

system *(HINI)*
Handle of the new system profile.

Returns

reply *(BIT32)*
Reserved.

NULL Reserved value, must be null.

Remarks

Applications should refresh their defaults from the user or system profile.

Default Processing

The default window procedure takes no action on this message, other than to set **reply** to NULL.

WM_ACTIVATE

This message occurs when an application causes the activation or deactivation of a window.

Parameters

param1

active *(BOOL)*
Active indicator:

TRUE The window is being activated
FALSE The window is being deactivated.

param2

hwnd *(HWND)*
Window handle.

In the case of activation, **hwnd** identifies the window being activated. In the case of deactivation, **hwnd** identifies the window being deactivated.

Returns

reply *(BIT32)*
Reserved.

NULL Reserved value.

Remarks

A deactivation message (that is, a WM_ACTIVATE message with **active** set to FALSE) is sent first to the window procedure of the main window being deactivated, before an activation message (that is, a WM_ACTIVATE message with **active** set to TRUE) is sent to the window procedure of the main window being activated.

Any WM_SETFOCUS messages with **focus** set to FALSE, are sent before the deactivation message. Any WM_SETFOCUS messages with **focus** set to TRUE, are sent after the activation message.

If WinSetFocus is called during the processing of a WM_ACTIVATE message, a WM_SETFOCUS message with **focus** set to FALSE is not sent, as no window has the focus.

If a window is activated before any of its children have the focus, this message is sent to the frame window or to its FID_CLIENT, if it exists.

Programming Note: Except in the instance of a WM_ACTIVATE message, with **active** set to TRUE, an application processing a WM_ACTIVATE or a WM_SETFOCUS message should not change the focus window or the active window. If it does, the focus and active windows must be restored before the window procedure returns from processing the message. For this reason, any dialog boxes or windows brought up during the processing of a WM_ACTIVATE or a WM_SETFOCUS message should be system modal.

Default Processing

The default window procedure takes no action on this message, other than to set **reply** to NULL.

WM_ADJUSTWINDOWPOS

This message is sent by the WinSetWindowPos call to enable the window to adjust its new position or size whenever it is about to be moved.

Parameters

param1

lpswp *(PSWP)*
SWP structure pointer.

The structure has been filled in by WinSetWindowPos with the proposed move/size data. The control can adjust this new position by changing the contents of the *SWP* structure. It can change the **x/y** fields to adjust its new position; it can change the **width/height** fields to adjust its new size, or it can change the **behind** field to adjust its new z-order.

param2

zero *(BIT32)*
Zero.

Returns

reply

result *(BIT32)*
Window-adjustment status indicators.

Bits 0 through 15 of this parameter are reserved for system use and bits 16 through 31 are available for application use.

NULL	No changes have been made.
AWP_MINIMIZED	The frame window has been minimized.
AWP_MAXIMIZED	The frame window has been maximized.
AWP_RESTORED	The frame window has been restored.
AWP_ACTIVATE	The frame window has been activated.
AWP_DEACTIVATE	The frame window has been deactivated.

Remarks

Frame controls can respond to this message to reposition themselves or resize themselves in the window frame.

Menu controls respond to this message as follows:

MS_ACTIONBAR not specified: The *SWP* **width** and *SWP* **height** fields are set so that the menu window exactly contains all of the items in the menu. The *SWP* **x** and *SWP* **y** fields are not changed.

MS_ACTIONBAR specified and MS_TITLEBUTTON not specified: The items in the menu are arranged such that all of the items are visible within the width specified by the *SWP* **width** field. This formatting may cause the menu items to be arranged in multiple lines. The *SWP* **width** field is set to include all of the lines of the menu. The *SWP* **x** and *SWP* **y** fields are not changed.

MS_ACTIONBAR specified and MS_TITLEBUTTON specified: The *SWP* **width** value is set to the accumulated width of the items in the menu. The height specified in the *SWP* **height** field is not changed. In both instances, the *SWP* **width** and *SWP* **height** fields are only altered if SWP_SIZE is specified in the **options** field. Instead, the width of MS_TITLEBUTTON menus is determined by the accumulated width of the items in the menu.

A list box does two things:

- Changes the height so as to accommodate an exact number of items.
- Automatically outsets its border. This means, for example, that the x, y, width, and height fields in the resource file specify the working area of the listbox. The border is drawn outside this area.

The entry field control, if ES_MARGIN is specified, outsets its margin. This means that in the resource file, the numbers specified as the x, y position of an entry field control are taken to be the position where the first character of text is drawn, not where the lower-left corner of the surrounding box is drawn. Similarly, the height and width parameters apply to the editable area of the control; consequently, they do not include the margin.

When a dialog is created with WinCreateDlg or WinLoadDlg, a WM_ADJUSTWINDOWPOS message is sent to each child window after the dialog window is created, with a pointer to a *SWP* structure containing **options** equal to SWP_SIZE | SWP_MOVE and the **x**, **y**, **height**, and **width** fields initialized to the window's current size and position. The message enables the control to adjust its size or position, usually to compensate for its border, or margin, or both.

Default Processing

The default window procedure takes no action on this message, other than to set **result** to NULL.

WM_APPTERMINATENOTIFY

This message is posted when an application terminates.

Parameters

param1

happ *(HAPP)*
Application handle.

param2

retcode *(BIT32)*
Return code from the terminating application.

Returns

reply *(BIT32)*
Reserved.

NULL Reserved value; must be null.

Remarks

The WinInstStartApp call provides the capability for the starting application to be notified when the started application terminates, by the use of the **NotifyWindow** parameter.

Default Processing

The default window procedure takes no action on this message, other than to set **reply** to NULL.

WM_BUTTON1DBLCLK

This message occurs when the operator presses button one of the pointing device twice within a specified time, as detailed below.

Parameters

param1

pointerpos *(POINTS)*
Pointer position.

The pointer position is in window coordinates relative to the bottom-left corner of the window.

param2

hittestres *(BIT16)*
Hit-test result.

hittestres provides the hit-test result. It contains the value returned from the hit-test process, which determines the window to be associated with this message. For details of the possible values, see "WM_HITTEST" on page 12-23.

Returns

reply

result *(BOOL)*
Processed indicator:

TRUE	Message processed
FALSE	Message ignored.

Remarks

This message is posted to the application queue associated with the window that is to receive the pointer-button information.

A double-click is recognized if all of the following are true:

- Two clicks are of the same button.
- No intervening pointing device button is pressed.
- The two clicks occur within the double-click time interval as defined by the SV_DBLCLKTIME system value.
- The two clicks occur within a small spatial distance. This is defined by the rectangle, the length of whose sides parallel to the x and y axes are respectively, the SV_CXDBLCLICK and SV_CYDBLCLICK system values. The first click is assumed to be at the center of this rectangle.

Default Processing

The default window procedure sends the message to the owner window if it exists, otherwise it takes no action on this message, other than to set **result** to FALSE.

WM_BUTTON1DOWN

This message occurs when the operator presses pointer button one.

Parameters

param1

pointerpos *(POINTS)*
Pointer position.

The pointer position is in window coordinates relative to the bottom-left corner of the window.

param2

hittestres *(BIT16)*
Hit-test result.

hittestres provides the hit-test result. It contains the value returned from the hit test process, which determined the window to be associated with this message. For details of the possible values, see "WM_HITTEST" on page 12-23.

Returns

reply

result *(BOOL)*
Processed indicator:

TRUE	Message processed
FALSE	Message ignored.

Remarks

This message is posted to the application queue associated with the window that is to receive the pointer-button information.

It is the responsibility of the application to ensure that the appropriate frame window is activated and that the focus is to the appropriate window, by using the WinSetFocus function.

Default Processing

The default window procedure activates the window using WinSetActiveWindow, and then sets **result** to FALSE.

WM_BUTTON1UP

This message occurs when the operator releases button one of the pointing device.

Parameters

param1

pointerpos *(POINTS)*
Pointer position.

The pointer position is in window coordinates relative to the bottom-left corner of the window.

param2

hittestres *(BIT16)*
Hit-test result.

hittestres provides the hit-test result. It contains the value returned from the hit-test process, which determines the window to be associated with this message. For details of the possible values, see "WM_HITTEST" on page 12-23.

Returns

reply

result *(BOOL)*
Processed indicator:

TRUE	Message processed
FALSE	Message ignored.

Remarks

This message is posted to the application queue associated with the window that is to receive the pointing device button information.

Default Processing

The default window procedure sends the message to the owner window if it exists, otherwise it takes no action on this message other than to set **result** to FALSE.

WM_BUTTON2DBLCLK

This message occurs when the operator presses button two of the pointing device twice within a specified time, as detailed in "WM_BUTTON1DBLCLK" on page 12-6.

Parameters

param1

pointerpos *(POINTS)*
Pointer position.

The pointer position is in window coordinates relative to the bottom-left corner of the window.

param2

hittestres *(BIT16)*
Hit-test result.

hittestres provides the hit-test result. It contains the value returned from the hit-test process, which determines the window to be associated with this message. For details of the possible values, see "WM_HITTEST" on page 12-23.

Returns

reply

result *(BOOL)*
Processed indicator:

TRUE	Message processed
FALSE	Message ignored.

Remarks

This message is posted to the application queue associated with the window that is to receive the pointer-button information.

Default Processing

The default window procedure processes this message identically to WM_BUTTON1DBLCLK.

WM_BUTTON2DOWN

This message occurs when the operator presses button two on the pointing device.

Parameters

param1

pointerpos *(POINTS)*
Pointer position.

The pointer position is in window coordinates relative to the bottom-left corner of the window.

param2

hittestres *(BIT16)*
Hit-test result.

hittestres provides the hit-test result. It contains the value returned from the hit test process, which determined the window to be associated with this message. For details of the possible values, see "WM_HITTEST" on page 12-23.

Returns

reply

result *(BOOL)*
Processed indicator:

TRUE	Message processed
FALSE	Message ignored.

Remarks

This message is posted to the application queue associated with the window that is to receive the pointing device button information.

It is the responsibility of the application to ensure that the appropriate frame window is activated and that the focus is to the appropriate window, by using the WinSetFocus function.

Default Processing

The default window procedure processes this message identically to "WM_BUTTON1DOWN" on page 12-7.

WM_BUTTON2UP

This message occurs when the operator releases button two of the pointing device.

Parameters

param1

pointerpos *(POINTS)*
Pointer position.

The pointer position is in window coordinates relative to the bottom-left corner of the window.

param2

hittestres *(BIT16)*
Hit-test result.

hittestres provides the hit-test result. It contains the value returned from the hit-test process, which determines the window to be associated with this message. For details of the possible values, see "WM_HITTEST" on page 12-23.

Returns

reply

result *(BOOL)*
Processed indicator:

TRUE	Message processed
FALSE	Message ignored.

Remarks

This message is posted to the application queue associated with the window that is to receive the pointing device button information.

Default Processing

The default window procedure sends the message to the owner window if it exists, otherwise it takes no action on this message other than to set **result** to FALSE.

WM_BUTTON3DBLCLK

This message occurs when the operator presses button three of the pointing device twice within a specified time, as detailed in "WM_BUTTON1DBLCLK" on page 12-6.

Parameters

param1

pointerpos *(POINTS)*
Pointer position.

The pointer position is in window coordinates relative to the bottom left corner of the window.

param2

hittestres *(BIT16)*
Hit-test result.

hittestres provides the hit-test result. It contains the value returned from the hit-test process, which determines the window to be associated with this message. For details of the possible values, see "WM_HITTEST" on page 12-23.

Returns

reply

result *(BOOL)*
Processed indicator:

TRUE	Message processed
FALSE	Message ignored.

Remarks

This message is posted to the application queue associated with the window that is to receive the pointer button information.

Default Processing

The default window procedure processes this message identically to WM_BUTTON1DBLCLK.

WM_BUTTON3DOWN

This message occurs when the operator presses button three on the pointing device.

Parameters

param1

pointerpos *(POINTS)*
Pointer position.

The pointer position is in window coordinates relative to the bottom-left corner of the window.

param2

hittestres *(BIT16)*
Hit-test result.

hittestres provides the hit-test result. It contains the value returned from the hit test process, which determined the window to be associated with this message. For details of the possible values, see "WM_HITTEST" on page 12-23.

Returns

reply

result *(BOOL)*
Processed indicator:

TRUE	Message processed
FALSE	Message ignored.

Remarks

This message is posted to the application queue associated with the window that is to receive the pointing device button information.

It is the responsibility of the application to ensure that the appropriate frame window is activated and that the focus is to the appropriate window, by using the WinSetFocus function.

Default Processing

The default window procedure processes this message identically to "WM_BUTTON1DOWN" on page 12-7.

WM_BUTTON3UP

This message occurs when the operator releases button three of the pointing device.

Parameters

param1

pointerpos *(POINTS)*
Pointer position.

The pointer position is in window coordinates relative to the bottom-left corner of the window.

param2

hittestres *(BIT16)*
Hit-test result.

hittestres provides the hit-test result. It contains the value returned from the hit-test process, which determines the window to be associated with this message. For details of the possible values, see "WM_HITTEST" on page 12-23.

Returns

reply

result *(BOOL)*
Processed indicator:

TRUE	Message processed
FALSE	Message ignored.

Remarks

This message is posted to the application queue associated with the window that is to receive the pointing device button information.

Default Processing

The default window procedure processes this message identically to WM_BUTTON1UP.

WM_CALCFRAMERECT

This message occurs when an application uses the WinCalcFrameRect call.

Parameters

param1

rect *(PRECTL)*
Rectangle structure.

This points to a *RECTL* structure.

param2

frame *(BOOL)*
Frame indicator:

TRUE	Frame rectangle provided
FALSE	Client area rectangle provided.

Returns

reply
Reserved.

success *(BOOL)*
Rectangle-calculated indicator:

TRUE	Successful completion
FALSE	Error occurred or the calculated rectangle is empty.

Remarks

This message is sent to Frame Control to perform the appropriate calculation.

Default Processing

The default window procedure takes no action on this message, other than to set **success** to FALSE.

WM_CALCVALIDRECTS

This message is sent from WinSetWindowPos and WinSetMultWindowPos to determine which areas of a window may be preserved if a window is sized, and which should be redisplayed.

Parameters

param1

oldnew *(PRECTL)*
Window-rectangle structures.

This points to two *RECTL* structures. The first structure contains the rectangle of the window before the move, the second contains the rectangle of the window after the move. The coordinates of the rectangles are relative to the parent window.

param2

new *(PSWP)*
New window position.

This points to a *SWP* structure (see the WinSetWindowPos call).

Returns

reply

align *(USHORT)*
Alignment control.

This instructs WinSetWindowPos how to align valid window bits. This value is made up from CVR_* flags, as follows:

CVR_ALIGNLEFT	Align with the left edge of the window.
CVR_ALIGNBOTTOM	Align with the bottom edge of the window.
CVR_ALIGNTOP	Align with the top edge of the window.
CVR_ALIGNRIGHT	Align with the right edge of the window.
CVR_REDRAW	The whole window is invalid. If CVR_REDRAW, is set, the whole window is assumed invalid, otherwise, the remaining flags can be ORed together to get different kinds of alignment. For example: `( CVR_ALIGNLEFT \| CVR_ALIGNTOP )` aligns the valid window area with the top-left of the window.
0	It is assumed the application has changed the rectangles pointed to by **oldnew** and **new** itself.

Remarks

This message is **not** sent if this window has the CS_SIZEREDRAW style, indicating size-sensitive window content that must be totally redrawn if sized.

This enables the application to determine if the window's position has changed as well as its size; this can aid alignment processing.

These rectangles can be modified by the window procedure to cause parts of the window to be redrawn and not preserved.

The window manager tries to preserve the screen image by copying the image described by the old rectangle into the image described by the new rectangle. In this way, an application can control the alignment of the preserved image as well, by changing the origin of the first rectangle.

If no change is made to either rectangle, the entire window area is preserved. If either rectangle is empty, the entire window area is completely redrawn by the operation.

Programming Note: This functionality can be used to optimize window updating when the window is resized. For example, if the application returns that the window is to be aligned with the top-left corner, and the top border is sized, the window's screen data moves with the top border.

In all instances, the rectangles are intersected with the area of the screen that is actually visible and the valid area of the window. That is, only the window area that contains window information is copied.

For example, consider an application that has two scroll bars, that are children of the client window. When the window is resized, the scroll bars must be completely redrawn. By returning rectangles that exclude the scroll bars, the area of the scroll bars is completely redrawn, thereby preserving only the part of the screen that is worth preserving.

Default Processing

The default window procedure processing is to align the valid area with the top-left of the window by returning:

```
( CVR_ALIGNTOP | CVR_ALIGNLEFT )
```

In addition, any child windows intersecting the source rectangle pointed to by **oldnew** of this message, are also offset with the aligned window area.

WM_CHAR

This message occurs when an operator presses a key.

Parameters

param1

flags *(BIT16)*
Keyboard control codes:

KC_CHAR	Indicates that **ch** value is valid.
KC_SCANCODE	Indicates that **scancode** is valid. Generally, this is set in all WM_CHAR messages generated from actual operator input. However, if the message has been generated by an application that has issued the WinSetHook function to filter key strokes, or posted to the application queue, this may not be set.
KC_VIRTUALKEY	Indicates that **vk** is valid. Normally **vk** should be given precedence when processing the message.
KC_KEYUP	The event is a key-up transition; otherwise it is a down transition.
KC_PREVDOWN	The key has been previously down; otherwise it has been previously up.
KC_DEADKEY	The character code is a dead key. The application is responsible for displaying the glyph for the dead key without advancing the cursor.
KC_COMPOSITE	The character code is formed by combining the current key with the previous dead key.
KC_INVALIDCOMP	The character code is not a valid combination with the preceding dead key. The application is responsible for advancing the cursor past the dead-key glyph and then, if the current character is not a space, sounding the alarm and displaying the new character code.
KC_LONEKEY	Indicates if the key is pressed and released without any other keys being pressed or released between the time the key goes down and up.
KC_SHIFT	The shift state is active when key press or release occurred.
KC_ALT	The ALT state is active when key press or release occurred.
KC_CTRL	The CTRL state was active when key press or release occurred.

repeat *(UCHAR)*
Repeat count.

scancode *(UCHAR)*
Hardware scan code.

A keyboard-generated value that identifies the keyboard event. This is the raw scancode, not the translated scancode.

param2

ch *(USHORT)*
Character code.

A character value translation of the keyboard event resulting from the current code page that would apply if the CTRL and ALT keys were not depressed.

vk *(USHORT)*
Virtual key codes.

A virtual key value translation of the keyboard event resulting from the virtual key code table. The low-order byte contains the **vk** value, and the high-order byte is always set to zero by the standard translate table.

0 This value applies if **flags** does not contain KC_VIRTUALKEY.

Returns

reply

result *(BOOL)*
Processed indicator:

TRUE Message processed
FALSE Message ignored.

Remarks

This message is posted to the queue associated with the window that has the focus.

The set of keys that causes a WM_CHAR message is device-dependent.

When this message is processed, precedence should normally be given to a valid virtual key if there is one contained in the message.

There are several instances when a window procedure may receive this message with the KC_KEYUP bit set, although it did not receive this message for the down transition of the key.

For example,

- The down transition of the key is translated by the function WinTranslateAccel, into a WM_COMMAND, WM_SYSCOMMAND, WM_HELP, or a WM_NULL message.
- The key down causes the input focus to change (tab to another window, dismiss a dialog, exit a program, and so on).
- Some other event happens that changes the focus between the time that the key is pressed down and the time that it is released.

Applications should normally only process WM_CHAR messages that do not have the KC_KEYUP bit set.

Except for the special instance where the LONEKEY flag is set on an accelerator key definition, all translations are done on the down stroke of the character.

When the current character is a double-byte character then **param2** contains both bytes of the double-byte character. These bytes are in the order CHAR1FROMMP, CHAR2FROMMP. When the current character is a single-byte character, CHAR2FROMMP contains NULL.

Default Processing

The default window procedure sends the message to the owner window if it exists, otherwise it takes no action on this message other than to set **result** to FALSE.

WM_CLOSE

This message is sent to a frame window to indicate that the window is being closed by the user.

Parameters

param1 *(BIT32)*
Reserved.

NULL Reserved value.

param2 *(BIT32)*
Reserved.

NULL Reserved value.

Returns

reply *(BIT32)*
Reserved.

NULL Reserved value.

Remarks

This message is sent by the frame to itself as a result of receiving a WM_SYSCOMMAND message with SC_CLOSE code set.

Default Processing

The default window procedure posts a WM_QUIT message to the appropriate queue and sets **reply** to NULL.

WM_COMMAND

This message occurs when a control has a significant event to notify to its owner or when a key stroke has been translated by an accelerator table into a WM_COMMAND.

Parameters

param1

cmd *(USHORT)*
Command value.

It is the application's responsibility to be able to relate **cmd** to an application function.

param2

source *(USHORT)*
Source type.

Identifies the type of control:

CMDSRC_PUSHBUTTON	Posted by a pushbutton control. **cmd** is the window identity of the pushbutton.
CMDSRC_MENU	Posted by a menu control. **cmd** is the identity of the menu item.
CMDSRC_ACCELERATOR	Posted as the result of an accelerator. **cmd** is the accelerator command value.
CMDSRC_OTHER	Other source. **cmd** gives further control-specific information defined for each control type.

pointer *(BOOL)*
Pointer-device indicator:

TRUE	The message is posted as a result of a pointer-device operation
FALSE	The message is posted as a result of a keyboard operation.

Returns

reply *(BIT32)*

Reserved.

NULL Reserved value.

Remarks

This message is posted to the queue of the owner of the control, thereby offering it the opportunity to perform some activity as a result.

Default Processing

The default window procedure takes no action on this message, other than to set **reply** to NULL.

WM_CONTROL

This message occurs when a control has a significant event to notify to its owner.

Parameters

param1

id *(IDENTITY)*

Control-window identity.

This is either the **id** parameter of the WinCreateWindow call or the identity of an item in a dialog template.

notifycode *(USHORT)*

Notify code.

The meaning of the notify code depends on the type of the control. For details, refer to the section describing that control.

param2

controlspec *(ULONG)*

Control-specific information.

The meaning of the control-specific information depends on the type of the control. For details, refer to the section describing that control.

Returns

reply *(BIT32)*

Reserved.

NULL Reserved value.

Remarks

This message is sent to the owner of the control, thereby offering it the opportunity to perform some activity before returning to the control.

Default Processing

The default window procedure takes no action on this message, other than to set **reply** to NULL.

WM_CONTROLHEAP

This message returns the heap handle to be used for allocating control-specific data.

Parameters

param1

cbheap *(ULONG)*

Initial heap allocation in bytes:

0 The heap is initialized to 512 bytes.

Other The heap is initialized with this number of bytes.

param2

hwndctl *(HWND)*
Window handle of the control.

Returns

reply

hheap *(HHEAP)*
Heap handle.

Remarks

This message is sent by menu, list box, and edit controls to their owner, to get the heap handle to use for allocating control-specific data. This message should be handled by returning a heap handle, that has been created with WinCreateHeap.

Default Processing

The default window procedure processes this message by returning the queue heap handle.

WM_CONTROLPOINTER

This message is sent to a control's owner window when the pointing device pointer moves over the control window, allowing the owner to set the pointing device pointer.

Parameters

param1

idctl *(USHORT)*
Control identifier.

param2

hptrnew *(HPOINTER)*
Handle of the pointing device pointer that the control is to use.

Returns

reply

hptrret *(HPOINTER)*
Returned pointing device-pointer handle that is then used by the control.

Remarks

The recommended approach for an application, that does not have specific reasons for controlling the pointer appearance, is to pass the message to the default window procedure.

Default Processing

The default window procedure returns **hptrnew**.

WM_CREATE

This message occurs when an application requests the creation of a window.

Parameters

param1

ctldata *(PCTLDATA)*
Control data.

This points to a *CTLDATA* data structure initialized with the data provided in the **CtlData** parameter of the WinCreateWindow call.

This pointer is also contained in the **create** parameter.

param2

create *(PCREATESTRUCT)*
Create structure.

This points to a *CREATESTRUCT* data structure.

Returns

reply

result *(BOOL)*
Error indicator:

TRUE Discontinue window creation
FALSE Continue window creation.

Remarks

This message is sent to the window procedure of the window being created, thus offering it an opportunity to initialize that window.

The window procedure receives this after the window is created but before the window becomes visible.

Default Processing

The default window procedure takes no action on this message, other than to set **result** to FALSE, which is equivalent to continuing the creation of the window.

WM_DESTROY

This message occurs when an application requests the destruction of a window.

Parameters

param1 *(BIT32)*
Reserved.

NULL Reserved value.

param2 *(BIT32)*
Reserved.

NULL Reserved value.

Returns

reply *(BIT32)*
Reserved.

NULL Reserved value.

Remarks

This message is sent to the window procedure of the window being destroyed after it has been hidden on the device, thereby offering it an opportunity to perform some termination action for that window.

Default Processing

The default window procedure takes no action on this message, other than to set **reply** to NULL.

WM_DRAWITEM

This notification is sent to the owner of a control each time an item is to be drawn.

Parameters

param1

identity *(IDENTITY)*
Window identifier.

The window identity of the control sending this notification message.

param2

controlspec *(ULONG)*
Control-specific information.

The meaning of the control-specific information depends on the type of control. For details of each control type, refer to the appropriate section.

Returns

reply

drawn *(BOOL)*
Item-drawn indicator:

TRUE The owner has drawn the item, and so the control does not draw it.
FALSE If the item contains text and the owner does not draw the item, the owner returns this value and the control draws the item.

Remarks

A control can only display some types of information, and emphasize items in a control-specific manner. Therefore, if special items are to be displayed or emphasized in a special manner, this must be done by the owner window of the control.

The control window procedure generates this message and sends it to the owner of the control, informing the owner that an item is to be drawn, offering the owner the opportunity to draw that item and to indicate that either the item has been drawn or that the control is to draw it.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set **drawn** to the default value of NULL, which is equivalent to FALSE.

WM_ENABLE

This message sets the enable state of a window.

Parameters

param1

newenabledstate *(BOOL)*
New enabled state indicator:

TRUE Set the window to enabled state
FALSE Set the window to disabled state.

param2 *(BIT32)*
Reserved.

NULL Reserved value.

Returns

reply *(BIT32)*
Reserved.

NULL Reserved value.

Remarks

This message is sent to the window procedure of the window whose enable state is changing, thereby offering it an opportunity to perform some action appropriate to new state of the window.

Default Processing

The default window procedure takes no action on this message, other than to set **reply** to NULL.

WM_ERROR

This message occurs when an error is detected in a WinGetMsg or a WinPeekMsg call.

Parameters

param1

errorcode *(USHORT)*
Error code.

param2 *(BIT32)*
Reserved.

NULL Reserved value.

Returns

reply *(BIT32)*
Reserved.

NULL Reserved value.

Remarks

The application can detect the error situation after the WinGetMsg or the WinPeekMsg call and before the WinDispatchMsg call.

Default Processing

The default window procedure takes no action on this message, other than to set **reply** to NULL.

WM_FOCUSCHANGE

For the cause of this message, see "WM_FOCUSCHANGE" on page 15-10.

Parameters

For a description of the parameters, see "WM_FOCUSCHANGE" on page 15-10.

Default Processing

The default window procedure sends this message to the owner or parent, if it exists and is not the desktop. Otherwise, it sets **reply** to NULL.

WM_FORMATFRAME

For the cause of this message, see "WM_FORMATFRAME" on page 15-11.

Parameters

For a description of the parameters, see "WM_FORMATFRAME" on page 15-11.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set **count** to the default value of NULL, which is equivalent to zero.

WM_HELP

This message occurs when a control has a significant event to notify to its owner or when a key stroke has been translated by an accelerator table into a WM_HELP.

Parameters

param1

cmd *(USHORT)*
Command value.

It is the application's responsibility to be able to relate **cmd** to an application function.

param2

source *(USHORT)*
Source type.

Identifies the type of control:

CMDSRC_PUSHBUTTON	Posted by a pushbutton control. **cmd** is the window identity of the pushbutton.
CMDSRC_MENU	Posted by a menu control. **cmd** is the identity of the menu item.
CMDSRC_ACCELERATOR	Posted as the result of an accelerator. **cmd** is the accelerator command value.
CMDSRC_OTHER	Other source. **cmd** gives further control-specific information defined for each control type.

pointer *(BOOL)*
Pointer-device indicator:

TRUE	If the message is posted as a result of a pointer-device operation
FALSE	If the message is posted as a result of a keyboard operation.

Returns

reply *(BIT32)*
Reserved.

NULL Reserved value.

Remarks

This message is identical to a WM_COMMAND message, but implies that the application should respond to this message by displaying help information.

This message is posted to the queue of the owner of the control.

Default Processing

The default window procedure sends this message to the parent window, if it exists and is not the desktop. Otherwise, it sets **reply** to NULL.

WM_HITTEST

This message is sent to determine which window is associated with an input from the pointing device.

Parameters

param1

pointerpos *(POINTS)*
Pointer position.

The pointer position is in window coordinates relative to the bottom-left corner of the window.

param2 *(BIT32)*
Reserved.

NULL Reserved value.

Returns

reply

result *(ULONG)*
Hit-test indicator.

The application may return one of these values:

HT_NORMAL	The message should be processed as normal. A WM_MOUSEMOVE, WM_BUTTON2DOWN, or WM_BUTTON1DOWN message is posted to the window.
HT_TRANSPARENT	The part of the window underneath the pointer is transparent; hit-testing should continue on windows underneath this window, as if the window did not exist.
HT_DISCARD	The message should be discarded; no message is posted to the application.
HT_ERROR	As HT_DISCARD, except that if the message is a button-down message, an alarm sounds.

Remarks

This message occurs when an application requests a message by issuing a WinPeekMsg or a WinGetMsg call.

If the message that is to be retrieved represents a pointer related event, this message is sent to a window to determine whether the message is in fact destined for that window.

This message is only sent if the window class has the CS_HITTEST style set.

Programming Note: The handling of this message determines whether a disabled window can process pointing device events.

Default Processing

The default window procedure takes no action on this message, other than to set **result** to HT_ERROR if the window is disabled, or to HT_NORMAL otherwise.

WM_HSCROLL

For the cause of this message, see "WM_HSCROLL" on page 20-2.

Parameters

For a description of the parameters, see "WM_HSCROLL" on page 20-2.

Default Processing

The default window procedure takes no action on this message, other than to set **reply** to NULL.

WM_INITDLG

For the cause of this message, see "WM_INITDLG" on page 12-51.

Parameters

For a description of the parameters, see "WM_INITDLG" on page 12-51.

Default Processing

The default window procedure takes no action on this message, other than to set **result** to FALSE.

WM_INITMENU

For the cause of this message, see "WM_INITMENU" on page 17-4.

Parameters

For a description of the parameters, see "WM_INITMENU" on page 17-4.

Default Processing

The default window procedure takes no action on this message, other than to set **reply** to NULL.

WM_JOURNALNOTIFY

This message is used to maintain correct operation during journal playback.

Parameters

param1

command *(ULONG)*
Command to journal.

JRN_QUEUESTATUS	The WinQueryQueueStatus command must be journaled.
JRN_PHYSKEYSTATE	The WinGetPhysKeyState command must be journaled.

param2
Data.

Data values depend on which command is to be journaled.

If **command** is set to JRN_QUEUESTATUS:

queuestatus *(BIT16)*
Queue status.

See the **Summary** parameter of the WinQueryQueueStatus call.

If **command** has the value JRN_PHYSKEYSTATE:

scancode *(USHORT)*
Scan code.

See the **Scancode** parameter of the WinGetPhysKeyState call.

keystate *(USHORT)*
Key State.

See the **KeyState** parameter of the WinGetPhysKeyState call.

Returns

reply *(BIT32)*

Reserved.

NULL Reserved value.

Remarks

If the WinQueryQueueStatus or the WinGetPhysKeyState calls have new information since the last time they were called and there is a journal record hook installed, the journal record hook is called with this message to record this new information.

During playback, this message is interpreted by the system and the appropriate state restored.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set **reply** to NULL.

WM_MATCHMNEMONIC

For the cause of this message, see "WM_MATCHMNEMONIC" on page 12-52.

Parameters

For a description of the parameters, see "WM_MATCHMNEMONIC" on page 12-52.

Default Processing

The default window procedure takes no action on this message, other than to set **result** to FALSE.

WM_MEASUREITEM

This notification is sent to the owner of a specific control to establish the height and width for an item in that control.

Parameters

param1

identity *(SHORT)*

Control identifier.

param2

controlspec *(ULONG)*

Control-specific information.

The meaning of the control-specific information depends on the type of control. For details of each control type, refer to the appropriate control section.

Returns

reply

height *(SHORT)*

Height of item.

width *(SHORT)*

Width of item.

Remarks

When the owner receives this message, it must calculate and return the height and width (for a horizontally-scrollable list box control) of an item to the control.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set **reply** to the default value of NULL, which is equivalent to zero.

WM_MENUEND

For the cause of this message, see "WM_MENUEND" on page 17-5.

Parameters

For a description of the parameters, see "WM_MENUEND" on page 17-5.

Default Processing

The default window procedure takes no action on this message, other than to set **reply** to NULL.

WM_MENUSELECT

For the cause of this message, see "WM_MENUSELECT" on page 17-6.

Parameters

For a description of the parameters, see "WM_MENUSELECT" on page 17-6.

Default Processing

The default window procedure takes no action on this message, other than to set **result** to TRUE.

WM_MINMAXFRAME

For the cause of this message, see "WM_MINMAXFRAME" on page 15-3.

Parameters

For a description of the parameters, see "WM_MINMAXFRAME" on page 15-3.

Default Processing

The default window procedure takes no action on this message, other than to set **overridedefault** to FALSE.

WM_MOUSEMOVE

This message occurs when the pointing device pointer moves.

Parameters

param1

This parameter contains the position of the pointing device in window coordinates relative to the bottom-left corner of the window.

xmouse *(SHORT)*
Pointing device x coordinate.

ymouse *(SHORT)*
Pointing device y coordinate.

param2

whittest *(USHORT)*
Message result:

Zero	A pointing device capture is currently in progress
Other	The result of the WM_HITTEST message.

Returns

reply

processed *(BOOL)*
Processed indicator:

TRUE The window procedure did process the message
FALSE The window procedure did not process the message.

Default Processing

The default window procedure sets the pointer shape using the WinSetPointer call and sets **processed** to FALSE.

WM_MOVE

This message occurs when a window with style CS_MOVENOTIFY changes its absolute position.

Parameters

param1 *(BIT32)*
Reserved.

NULL Reserved value.

param2 *(BIT32)*
Reserved.

NULL Reserved value.

Returns

reply *(BIT32)*
Reserved.

NULL Reserved value.

Remarks

The message is sent from WinSetWindowPos, WinSetMultWindowPos, and WinScrollWindow.

The message is sent to any window when it is moved relative to its parent window. In addition, a WM_MOVE message is also sent to any children of that window that have style CS_MOVENOTIFY.

The new position of the window is obtained by calling WinQueryWindowRect, and can make those rectangle coordinates relative to any window by calling WinMapWindowPoints.

Programming Note: There are several instances where windows need to know if they have been moved, and these include the occasions when the window does not change position relative to its parent, but does change position relative to the screen (its absolute position).

An example is menus. When a top-level menu control (child of the frame window) moves its absolute position as a result of the frame window being moved, the top-level menu control needs to move any pull down menus along with its movement. The same applies to application/dialog box positional grouping. In some instances, a dialog box might need to be moved as the main window is moved, to make room for other applications.

Default Processing

The default window procedure takes no action on this message, other than to set **reply** to NULL.

WM_NEXTMENU

For the cause of this message, see "WM_NEXTMENU" on page 17-7.

Parameters

For a description of the parameters, see "WM_NEXTMENU" on page 17-7.

Default Processing

The default window procedure takes no action on this message, other than to set **newmenu** to NULL.

WM_NULL

This message is not sent by the system.

Parameters

param1 *(BIT32)*
Reserved.

NULL Reserved value.

param2 *(BIT32)*
Reserved.

NULL Reserved value.

Returns

reply *(BIT32)*
Reserved.

NULL Reserved value; must be NULL.

Remarks

This message does not do anything.

Default Processing

The default window procedure takes no action on this message, other than to set **reply** to NULL.

WM_OTHERWINDOWDESTROYED

This message is sent to all top-level windows when a registered window is being destroyed.

Parameters

param1

destroyed *(HWND)*
Handle of destroyed window.

param2 *(BIT32)*
Reserved.

NULL Reserved value.

Returns

reply *(BIT32)*
Reserved.

NULL Reserved value; must be NULL.

Remarks

This message is sent by the WinDestroyWindow function after the window, which was registered by use of the WinRegisterWindowDestroy call, is hidden, but before it is destroyed.

Default Processing

The default window procedure takes no action on this message, other than to set **reply** to NULL.

WM_PACTIVATE

This message is posted when the Language Support Window or Dialog Procedure processes a WM_ACTIVATE message.

Parameters

param1

active *(BOOL)*
Active indicator:

TRUE The window was activated
FALSE The window was deactivated.

param2

hwnd *(HWND)*
Window handle.

In the case of activation, **hwnd** identifies the window which was activated. In the case of deactivation, **hwnd** identifies the window which was deactivated.

Returns

reply *(BIT32)*
Reserved.

NULL Reserved value.

Remarks

The activation change has already occurred when the application receives this message.

Default Processing

The default window procedure takes no action on this message, other than to set **reply** to NULL.

WM_PCONTROL

This message is posted when the Language Support Window or Dialog Procedure processes a WM_CONTROL message.

Parameters

param1

id *(IDENTITY)*
Control-window identity.

This is either the **id** parameter of the WinCreateWindow call or the identity of an item in a dialog template.

notifycode *(USHORT)*
Notify code.

The meaning of the notify code depends on the type of the control. For details, refer to the section describing that control.

param2 *(BIT32)*
Null.

NULL The control-specific information in **controlspec** of the WM_CONTROL message is not available because the information may not be valid when the application receives this message.

Returns

reply *(BIT32)*
Reserved.

NULL Reserved value.

Remarks

The notification from the control has already been processed when the application receives this message.

Default Processing

The default window procedure takes no action on this message, other than to set **reply** to NULL.

WM_PAINT

This message occurs when a window needs repainting.

Parameters

param1 *(BIT32)*
Reserved.

NULL Reserved value.

param2 *(BIT32)*
Reserved.

NULL Reserved value.

Returns

reply *(BIT32)*
Reserved.

NULL Reserved value.

Default Processing

The default window procedure issues the WinBeginPaint and WinEndPaint calls, and then sets **reply** to NULL.

WM_PPAINT

This message is posted when the Language Support Window or Dialog Procedure processes a WM_PAINT message.

Parameters

param1 *(BIT32)*
Reserved.

NULL Reserved value.

param2 *(BIT32)*
Reserved.

NULL Reserved value.

Returns

reply *(BIT32)*
Reserved.

NULL Reserved value.

Default Processing

The default window procedure issues the WinBeginPaint and WinEndPaint calls, and then sets **reply** to NULL.

WM_PRESPARAMCHANGED

This message is sent when a presentation parameter is set or removed dynamically from a window instance using the WinSetPresParam or WinRemovePresParam calls. It is also sent to all windows owned by the window whose presentation parameter was changed.

Parameters

param1

attrtype *(IDENTITY4)*
Presentation parameter attribute identity.

param2 *(BIT32)*
Reserved.

NULL Reserved value.

Returns

reply *(BIT32)*
Reserved.

NULL Reserved value, must be null.

Remarks

This is a notification message so that controls using a given inherited presentation parameter can find out when it changes.

Default Processing

The default window procedure takes no action on this message, other than to set **reply** to NULL.

WM_PSETFOCUS

This message is posted when the Language Support Window or Dialog Procedure processes a WM_SETFOCUS message.

Parameters

param1

hwnd *(HWND)*
Focus-window handle:

NULL No window lost or received the focus
Other Window handle.

param2

focus *(BOOL)*
Focus flag:

TRUE The window received the focus. **hwnd** is the window handle of the window which lost the focus, or NULL if no window previously had the focus.
FALSE The window lost the focus. **hwnd** is the window handle of the window which received the focus, or NULL if no window received the focus.

Returns

reply *(BIT32)*
Reserved.

NULL Reserved value.

Remarks

The focus change has already occurred when the application receives this message.

Default Processing

The default window procedure takes no action on this message, other than to set **reply** to NULL.

WM_PSIZE

This message is posted when the Language Support Window or Dialog Procedure processes a WM_SIZE message.

Parameters

param1

cxold *(SHORT)*
Old horizontal size.

cyold *(SHORT)*
Old vertical size.

param2

cxnew *(SHORT)*
New horizontal size.

cynew *(SHORT)*
New vertical size.

Returns

reply *(BIT32)*
Reserved.

NULL Reserved value.

Remarks

The size change has already occurred when the application receives this message.

Default Processing

The default window procedure takes no action on this message, other than to set **reply** to NULL.

WM_PSYSCOLORCHANGE

This message is posted when the Language Support Window or Dialog Procedure processes a WM_SYSCOLORCHANGE message.

Parameters

param1

options *(BIT32)*
Options.

Copied from the **Options** parameter of the WinSetSysColors function.

param2 *(BIT32)*
Reserved.

NULL Reserved value.

Returns

reply *(BIT32)*
Reserved.

NULL Reserved value.

Default Processing

The default window procedure takes no action on this message, other than to set **reply** to NULL.

WM_QUERYACCELTABLE

This message returns the handle to a window's accelerator table.

Parameters

param1 *(BIT32)*
Reserved.

NULL Reserved value.

param2 *(BIT32)*
Reserved.

NULL Reserved value.

Returns

reply

haccel *(HACCEL)*
Accelerator table handle:

NULL No accelerator table is associated with the window.
Other The handle of the accelerator table associated with the window.

Default Processing

The default window procedure takes no action on this message, other than to set **haccel** to NULL.

WM_QUERYCONVERTPOS

This message is sent by an application to determine whether it is appropriate to begin conversion of DBCS characters.

Parameters

param1

cursorpos *(PRECTL)*
Cursor position.

If **code** = QCP_CONVERT, **cursorpos** should be updated to contain the position of the cursor in the window receiving this message. The position is specified as a rectangle in screen coordinates.

If **code** = QCP_NOCONVERT, **cursorpos** should not be updated.

param2 *(BIT32)*
Reserved.

NULL Reserved value.

Returns

reply

code *(USHORT)*
Conversion code.

QCP_CONVERT Conversion may be performed for the window with the input focus, **cursorpos** has been updated to contain the position of the cursor.

QCP_NOCONVERT Conversion should not be performed, the window with the input focus cannot receive DBCS characters, **cursorpos** has not been updated.

Remarks

This message enables a DBCS application to determine whether the window with the input focus can handle DBCS characters. The **cursorpos** parameter can be used as a guide for positioning any conversion window that the application requires.

Default Processing

The default window procedure returns QCP_CONVERT, and updates **cursorpos** to the following values:

xleft = −1
ybottom = −1
xright = 0
ytop = 0

WM_QUERYTRACKINFO

The frame control and title bar control generate this message on receiving a WM_TRACKFRAME message or a TBM_TRACKMOVE message respectively.

Parameters

param1

tflags *(BIT16)*
Tracking flags.

Contains a combination of one or more TF_* flags as defined in the *TRACKINFO* structure.

param2

trackinfo *(PTRACKINFO)*
Track information structure.

This points to a *TRACKINFO* structure. The receiver of this message must modify this structure.

Returns

reply

result *(BOOL)*
Continue indicator:

TRUE Continue sizing or moving
FALSE Terminate sizing or moving.

Remarks

This message is sent to the window procedure of the owner of a frame control or title bar control respectively.

The *TRACKINFO* data structure specified by the **trackinfo** parameter is not initialized before the message is sent. It must be correctly completed before returning.

Default Processing

The default window procedure takes no action on this message, other than to set **result** to FALSE.

WM_QUERYWINDOWPARAMS

This message occurs when an application queries the window parameters.

Parameters

param1

wndparams *(PWNDPARAMS)*
Window parameter structure.

This points to a window parameter structure; see WNDPARAMS on page 2-35.

The valid values of **status** are WPM_CCHTEXT, WPM_TEXT, WPM_CBCTLDATA, and WPM_CTLDATA.

The flags in **status** are cleared as each item is processed. If the call is successful, **status** is NULL. If any item has not been processed, the flag for that item is still set.

param2 *(BIT32)*
Reserved.

NULL Reserved value.

Returns

reply

result *(BOOL)*
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Remarks

If this message is sent to a window of another process, the information in, or identified by, **wndparams** must be in memory shared by both processes.

Default Processing

The default window procedure sets the **length**, **presparamslength**, and **ctldatalength** parameters of the *WNDPARAMS* data structure identified by **wndparams** to zero, and sets **result** to FALSE.

WM_QUIT

This message is posted to terminate the application.

Parameters

param1 *(BIT32)*
Reserved.

NULL Reserved value.

param2 *(BIT32)*
Reserved.

NULL Reserved value.

Returns

reply *(BIT32)*
Reserved.

NULL Reserved value.

Remarks

It causes WinGetMsg to return **Result** set to FALSE, rather than to TRUE, as for all other messages.

Programming Note: Applications that call WinPeekMsg rather than WinGetMsg should test explicitly for WM_QUIT.

This message should not be dispatched to the default window procedure. The intent of this message is to cause the WinGetMsg loop to terminate.

Typically this message is posted by the application when the application exit command is selected from the action bar.

This message is also sent to all applications when the system is closing down. To reply to this, the application should either cancel the request by issuing an WinCancelShutdown call or close itself down by issuing a WinDestroyMsgQueue call.

Default Processing

The default window procedure takes no action on this message, other than to set **reply** to NULL.

WM_SAVEAPPLICATION

This message is sent by the system to notify an application to save its current state.

Parameters

param1 *(BIT32)*
Reserved.

NULL Reserved value.

param2 *(BIT32)*
Reserved.

NULL Reserved value.

Returns

reply *(BIT32)*
Reserved.

NULL Reserved value; must be NULL.

Remarks

When an application receives this message, it is expected to save its current state by any convenient method, for example, in a profile or in an auxiliary file.

It is the responsibility of the application to use the saved information, as appropriate, when it is resumed.

Even if the application processes this message, it should also pass it to the default window procedure, by using the WinDefWindowProc call.

Default Processing

The default window procedure takes no action on this message, other than to set **reply** to NULL.

WM_SEM1

This message is sent or posted by an application.

Parameters

accumbits *(BIT32)*
Semaphore value.

The semaphore values from all the WM_SEM1 messages posted to a queue, are accumulated by a logical-OR operation.

param2 *(BIT32)*
Reserved.

NULL Reserved value.

Returns

reply *(BIT32)*
Reserved.

NULL Reserved value.

Remarks

If the message is posted, it is merged with any existing WM_SEM1 message on the queue by combining the two **accumbits** values using a logical-OR operation.

The WM_SEM1 messages are queued higher than any other type of message.

Default Processing

The default window procedure takes no action on this message, other than to set **reply** to NULL.

WM_SEM2

This message is sent or posted by an application.

Parameters

accumbits *(BIT32)*
Semaphore value.

The semaphore values from all the WM_SEM2 messages posted to a queue, are accumulated by a logical-OR operation.

param2 *(BIT32)*
Reserved.

NULL Reserved value.

Returns

reply *(BIT32)*
Reserved.

NULL Reserved value.

Remarks

If the message is posted, it is merged with any existing WM_SEM2 message on the queue by combining the two **accumbits** values using a logical-OR operation.

The WM_SEM2 messages are queued above WM_SEM3 and WM_SEM4 messages, and above any WM_PAINT or WM_TIMER messages generated by the system, but lower than any other message.

Default Processing

The default window procedure takes no action on this message, other than to set **reply** to NULL.

WM_SEM3

This message is sent or posted by an application.

Parameters

accumbits *(BIT32)*

Semaphore value.

The semaphore values from all the WM_SEM3 messages posted to a queue, are accumulated by a logical-OR operation.

param2 *(BIT32)*

Reserved.

NULL Reserved value.

Returns

reply *(BIT32)*

Reserved.

NULL Reserved value.

Remarks

If the message is posted, it is merged with any existing WM_SEM3 message on the queue by combining the two **accumbits** values using a logical-OR operation.

The WM_SEM3 messages are queued above WM_SEM4 messages, and any WM_PAINT messages generated by the system, but lower than any other message.

Default Processing

The default window procedure takes no action on this message, other than to set **reply** to NULL.

WM_SEM4

This message is sent or posted by an application.

Parameters

accumbits *(BIT32)*

Semaphore value.

The semaphore values from all the WM_SEM4 messages posted to a queue, are accumulated by a logical-OR operation.

param2 *(BIT32)*

Reserved.

NULL Reserved value.

Returns

reply *(BIT32)*

Reserved.

NULL Reserved value.

Remarks

If the message is posted, it is merged with any existing WM_SEM4 message on the queue by combining the two **accumbits** values using a logical-OR operation.

The WM_SEM4 messages are queued lower than any other type of message.

Default Processing

The default window procedure takes no action on this message, other than to set **reply** to NULL.

WM_SETACCELTABLE

This message establishes the window accelerator table to be used for translation, when the window is active.

Parameters

param1

haccelnew *(HACCEL)*
New accelerator table.

param2 *(BIT32)*
Reserved.

NULL Reserved value.

Returns

reply

success *(BOOL)*
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Default Processing

The default window procedure takes no action on this message, other than to set **success** to FALSE.

WM_SETFOCUS

This message occurs when a window is to receive or lose the input focus.

Parameters

param1

hwnd *(HWND)*
Focus-window handle:

NULL No window is losing or receiving the focus.
Other Window handle.

param2

focus *(BOOL)*
Focus flag:

TRUE The window is receiving the focus. **hwnd** is the window handle of the window losing the focus, or NULL if no window previously had the focus.
FALSE The window is losing the focus. **hwnd** is the window handle of the window receiving the focus, or NULL if no window is receiving the focus.

Returns

reply *(BIT32)*

Reserved.

NULL Reserved value.

Remarks

This message is sent to the window receiving or losing the focus, thereby giving it the opportunity to perform some appropriate processing.

Programming Note: Except in the instance of WM_ACTIVATE, with **active** set to TRUE, an application processing WM_SETFOCUS or WM_ACTIVATE messages should not change the focus window or active window. If it does, the focus and active window must be restored before the application returns from processing the message. For this reason, any dialog boxes or windows brought up during the processing of WM_SETFOCUS or WM_ACTIVATE messages should be system modal.

Default Processing

The default window procedure takes no action on this message, other than to set **reply** to NULL.

WM_SETSELECTION

This message occurs when a window is selected or deselected.

Parameters

param1

selection *(BOOL)*

Selection flag:

TRUE The window is selected
FALSE The window is deselected.

param2 *(BIT32)*

Reserved.

NULL Reserved value.

Returns

reply *(BIT32)*

Reserved.

NULL Reserved value.

Remarks

The window procedure is expected to highlight or unhighlight the selected item of the window, as appropriate.

Default Processing

The default window procedure takes no action on this message, other than to set **reply** to NULL.

WM_SETWINDOWPARAMS

This message occurs when an application sets or changes the window parameters.

Parameters

param1

wndparams *(PWNDPARAMS)*

Window parameter structure.

This points to a window parameter structure; see WNDPARAMS on page 2-35.

The valid values of **status** are WPM_TEXT and WPM_CTLDATA.

param2 *(BIT32)*
Reserved.

NULL Reserved value.

Returns

reply

result *(BOOL)*
Success indicator:

TRUE Successful operation
FALSE Error occurred.

Remarks

If this message is sent to a window of another process, the information in, or identified by, **wndparams** must be in memory shared by both processes.

Default Processing

The default window procedure takes no action on this message, other than to set **result** to FALSE.

WM_SHOW

This message occurs when a window's WS_VISIBLE state is being changed.

Parameters

param1

show *(BOOL)*
Show indicator:

TRUE Show the window
FALSE Hide the window.

param2 *(BIT32)*
Reserved.

NULL Reserved value.

Returns

reply *(BIT32)*
Reserved.

NULL Reserved value.

Remarks

The message is sent after the visibility state has changed.

In this context, the terms "shown" or "hidden" refer to the state of the WS_VISIBLE style bit. This message is **not** sent when a window is obscured by other windows above it.

Default Processing

The default window procedure takes no action on this message, other than to set **reply** to NULL.

WM_SIZE

This message occurs when a window changes its size.

Parameters

param1

cxold *(SHORT)*
Old horizontal size.

cyold *(SHORT)*
Old vertical size.

param2

cxnew *(SHORT)*
New horizontal size.

cynew *(SHORT)*
New vertical size.

Returns

reply *(BIT32)*
Reserved.

NULL Reserved value.

Remarks

This message is not sent by WinCreateWindow when a window is created, and so any size-related processing must be done during the WM_CREATE message processing in this instance.

This message is sent after the window has been actually sized, but before any repainting has been done. Any resizing or repositioning of child windows that may be necessary as a result of the size change is usually done during the processing of this message.

Programming Note: It is generally unwise to output to the window during the processing of this message, because the area drawn may be redrawn, after the WM_SIZE processing is complete, by the WinSetWindowPos call.

The processing of this message for a window which is displaying an advanced VIO presentation space must be carried out by the default advanced VIO window procedure.

Default Processing

The default window procedure takes no action on this message, other than to set **reply** to NULL.

WM_SUBSTITUTESTRING

This message is sent from the WinSubstituteStrings call.

Parameters

param1

index *(INDEX2)*
Substitution index.

A value corresponding to the decimal character in the substitution phrase.

param2 *(BIT32)*
Reserved.

NULL Reserved value.

Returns

reply

string *(PSTRL)*
String to be substituted:

This points to a *STRL*.

NULL No substitution string
Other Substitution string.

Remarks

The WinSubstituteStrings call has encountered a substitution phrase in a string. The substitution phrase takes the form '%<digit>', where <digit> is a single decimal character; that is, 0 through 9.

Default Processing

The default window procedure takes no action on this message, other than to set **reply** to NULL.

WM_SYSCOLORCHANGE

This message is sent to all main windows when a change is made to the system colors by the WinSetSysColors function.

Parameters

param1

options *(BIT32)*
Options.

Copied from the **Options** parameter of the WinSetSysColors function and therefore specifies which palette has been changed.

param2 *(BIT32)*
Reserved.

NULL Reserved value.

Returns

reply *(BIT32)*
Reserved.

NULL Reserved value.

Remarks

All windows are invalidated, so that they are redrawn with the new colors. When this message is received, applications that depend on the system colors can query the new color values with the WinQuerySysColor call.

Default Processing

The default window procedure takes no action on this message, other than to set **reply** to NULL.

WM_SYSCOMMAND

This message occurs when a control has a significant event to notify to its owner or when a key stroke has been translated by an accelerator table into a WM_SYSCOMMAND message.

Parameters

param1

cmd *(USHORT)*
Window identifier.

It is the application's responsibility to be able to relate **cmd** to an application function.

param2

source *(USHORT)*
Source type.

Identifies the type of control:

CMDSRC_PUSHBUTTON	Posted by a pushbutton control. **cmd** is the window identifier of the pushbutton.
CMDSRC_MENU	Posted by a menu control. **cmd** is the identifier of the menu item.
CMDSRC_ACCELERATOR	Posted as the result of an accelerator. **cmd** is the accelerator command value.
CMDSRC_OTHER	Other source. **cmd** gives further control-specific information defined for each control type.

pointer *(BOOL)*
Pointing-device indicator:

TRUE	The message is posted as a result of a pointing-device operation
FALSE	The message is posted as a result of a keyboard operation.

Returns

reply *(BIT32)*
Reserved.

NULL	Reserved value.

Remarks

This message is posted to the queue of the owner of the control, thereby offering it the opportunity to perform some activity as a result.

Default Processing

The default window procedure takes no action on this message, other than to set **reply** to NULL.

WM_SYSVALUECHANGED

This message is posted to all main windows when one of the settable system values is changed.

Parameters

param1

changedfirst *(USHORT)*
First system value.

The first of a contiguous set of system values that has been changed.

param2

changedlast *(USHORT)*
Last system value.

The last of a contiguous set of system values that has been changed.

Returns

reply *(BIT32)*
Reserved.

NULL	Reserved value.

Remarks

If **changedfirst** equals **changedlast**, only one system value has changed.

If an application changes the settable system values, it is the application's responsibility to post this message to all main windows.

This message is processed by WC_FRAME windows by doing any frame-specific processing (such as sending WM_SETBORDERSIZE messages to the size border if SV_CX/CYSIZEBORDER system values have changed) and then sending the message to the client window if one exists.

This message is only posted when settable system values change.

Default Processing

The default window procedure takes no action on this message, other than to set **reply** to NULL.

WM_TIMER

This message is posted when a timer times out.

Parameters

param1

timer *(IDENTITY)*
Timer identity.

param2 *(BIT32)*
Reserved.

NULL Reserved value.

Returns

reply *(BIT32)*
Reserved.

NULL Reserved value.

Remarks

This message is always queued and is processed specially by the WinGetMsg and WinPeekMsg calls, as follows:

1. Timers are processed only by the WinGetMsg and WinPeekMsg calls.
2. A timer posts only one WM_TIMER message at a time.
3. WM_TIMER messages are queued lower than all other messages except WM_SEM3, WM_PAINT, and WM_SEM4 messages.

Default Processing

The default window procedure takes no action on this message, other than to set **reply** to NULL.

WM_TRACKFRAME

This message is sent to a window whenever it is to be moved or sized.

Parameters

param1

trackflags *(BIT16)*
Tracking flags.

Contains a combination of one or more TF_* flags; for details, see the *TRACKINFO* data structure.

param2 *(BIT32)*
Reserved.

NULL Reserved value.

Returns

reply

result *(BOOL)*
Success indicator:

TRUE The operation is successful
FALSE The operation is unsuccessful, or the operation is terminated.

Remarks

Respond to this message by causing a tracking rectangle to be drawn to move or size the window. For information, see WinTrackRect.

Default Processing

None.

WM_TRANSLATEACCEL

This message is sent to the focus window whenever a WM_CHAR message occurs.

Parameters

param1

qmsg *(PQMSG)*
QMSG structure.

This points to a QMSG structure.

param2 *(BIT32)*
Reserved.

NULL Reserved value.

Returns

reply

translated *(BOOL)*
Translated indicator:

TRUE The character exists in the accelerator table and has been translated in the *QMSG* structure

FALSE The character does not exist in the accelerator table or the window does not have an accelerator table.

Remarks

Normally this message is not processed by the focus window, but is passed to its parent, which passes it to its parent, until a frame window is reached.

Default Processing

The default window procedure takes no action on this message, other than to set **translated** to FALSE.

WM_TRANSLATEMNEMONIC

This message occurs during frame control processing of a WM_TRANSLATEACCEL message.

Parameters

param1

qmsg *(PQMSG)*
QMSG structure.

This points to a *QMSG* structure.

param2 *(BIT32)*
Reserved.

NULL Reserved value.

Returns

reply

success *(BOOL)*
Success indicator:

TRUE The character has been translated into an accelerator
FALSE The character has not been translated into an accelerator.

Remarks

This message is sent by the frame control to itself during the processing of a WM_TRANSLATEACCEL message, if the frame control does not translate a character into an accelerator by use of the frame window or queue accelerator tables.

When the frame control receives this message, it sends it to the application menu window, that is the window with identity FID_MENU.

Default Processing

The default window procedure takes no action on this message, other than to set **success** to FALSE.

WM_UPDATEFRAME

For the cause of this message, see "WM_UPDATEFRAME" on page 15-20.

Parameters

For a description of the parameters, see "WM_UPDATEFRAME" on page 15-20.

Default Processing

The default window procedure takes no action on this message, other than to set **result** to FALSE.

WM_VSCROLL

For the cause of this message, see "WM_VSCROLL" on page 20-3.

Parameters

For a description of the parameters, see "WM_VSCROLL" on page 20-3.

Default Processing

The default window procedure takes no action on this message, other than to set **reply** to NULL.

WM_WINDOWPOSCHANGED

This message is sent to the window procedure of the window whose position is changed.

Parameters

param1

swp *(PSWP)*
SWP structures.

This points to two *SWP* structures. The first *SWP* structure describes the entire new window state, whereas the second structure describes the entire old window state. The **options** parameter of the first structure contains only those indicators corresponding to the state changes that occurred.

param2

awp *(BIT32)*
Adjust window position status indicators.

The return value from the WM_ADJUSTWINDOWPOS message:

0 The SWP_NOADJUST option has been specified.
Other Adjust window position status indicators.

The AWF_* flags specify the state change of the frame window.

Returns

reply *(BIT32)*
Reserved.

NULL Reserved value.

Default Processing

The default window procedure sets **reply** to NULL and sends the following messages, based on the values of the **options** parameter of the first *SWP* data structure:

SWP_SIZE WM_SIZE with the new window size from the first *SWP* structure
SWP_HIDE WM_SHOW to hide the new window
SWP_SHOW WM_SHOW to show the new window.

Default Dialog Processing

This section describes how messages are processed by the default dialog procedure. The default dialog procedure can be called using WinDefDlgProc. A user dialog procedure should make this call for all messages that it does not want to process.

For WM_* messages other than those specified in this section the Default Dialog Procedure takes the same action and sets **result** to the same value as in Chapter 15, "Frame Control Window Processing." In the instance of messages that would be sent to FID_CLIENT, they are passed to the default window procedure.

For any other messages the default window procedure takes no action, other than to set **reply** to NULL.

WM_CHAR

For the cause of this message, see "WM_CHAR" on page 12-14.

Parameters

For a description of the parameters, see "WM_CHAR" on page 12-14.

Default Processing

If KC_CHAR is the mnemonic for a button that already has the focus, a BM_CLICK is sent to that button and **result** is set to TRUE. If the button does not have the focus, it receives the focus and **result** is set to TRUE.

If **vk** contains the value VK_TAB, the focus is set to the next tab item in the dialog. **result** is set to TRUE.

If **vk** contains the value VK_BACKTAB, the focus is set to the previous tab item in the dialog. **result** is set to TRUE.

If **vk** contains the value VK_LEFT or VK_UP, the focus is set to the previous item in the group. **result** is set to TRUE.

If **vk** contains the value VK_RIGHT or VK_BOTTOM, the focus is set to the next item in the group. **result** is set to TRUE.

If **vk** contains the value VK_ENTER or VK_NEWLINE, if a pushbutton has the focus a BM_CLICK is sent to that button. **result** is set to TRUE. If another control in the dialog has the focus the dialog is searched for a pushbutton with style BS_DEFAULT. If a pushbutton of this style is found, a BM_CLICK is sent to that button and **result** is set to TRUE.

If **vk** contains the value VK_ESC, WM_COMMAND is posted, with **source** is set to CMDSRC_PUSHBUTTON and **cmd** is set to DID_CANCEL. **result** is set to TRUE.

In other instances, if an owner exists the message is sent to the owner, otherwise **result** is set to FALSE.

WM_CLOSE

For the cause of this message, see "WM_CLOSE" on page 12-16.

Parameters

For a description of the parameters, see "WM_CLOSE" on page 12-16.

Default Processing

The default dialog procedure responds to this message by dismissing the dialog by issuing the WinDismissDlg call with its **Result** parameter set to DID_CANCEL.

WM_COMMAND

For the cause of this message, see "WM_COMMAND" on page 12-16.

Parameters

For a description of the parameters, see "WM_COMMAND" on page 12-16.

Default Processing

The default dialog procedure responds to this message by dismissing the dialog and passing **cmd** (the control item identifier) as **Result** of the WinProcessDlg or the WinDlgBox call that initiated the dialog and by setting **reply** to NULL.

WM_INITDLG

This message occurs when a dialog box is being created.

Parameters

param1

hwnd *(HWND)*
Focus window handle.

The handle of the control window that is to receive the input focus.

param2

create *(PCREATEPARAMS)*
Application-defined data area.

This points to the data area and is passed by the WinLoadDlg, WinCreateDlg, and WinDlgBox functions in their **CreateParams** parameter.

Returns

reply

result *(BOOL)*
Focus set indicator:

TRUE Focus window is changed. The dialog procedure can change the window to receive the focus, by issuing a WinSetFocus whose **NewFocus** specifies the handle of another control within the dialog box.
FALSE Focus window is not changed.

Remarks

This message is sent to the dialog procedure, before the dialog box is shown, thereby offering the dialog procedure the opportunity to perform initialization of the dialog box.

If any string substitutions are made by WinSubstituteStrings when the dialog is created, the WM_SUBSTITUTESTRING message may have been sent before the WM_INITDLG message is sent.

Default Processing

The default dialog procedure passes this message to the default window procedure, which sets **result** to FALSE.

WM_MATCHMNEMONIC

This message is sent by the dialog box to a control window to determine whether a typed character matches a mnemonic in its window text.

Parameters

param1

match *(USHORT)*
Match character.

param2 *(BIT32)*
Reserved.

NULL Reserved value.

Returns

reply

result *(BOOL)*
Match indicator:

TRUE Mnemonic found
FALSE Mnemonic not found, or an error occurred.

Remarks

This message is only processed by Button and Static Controls; all other controls return FALSE.

Default Processing

The default dialog procedure takes no action on this message, other than to set **result** to FALSE.

WM_QUERYDLGCODE

This message is sent by the dialog manager to identify the type of control, to determine what kinds of messages the control understands, and also to determine whether an input message may be processed by the dialog manager or passed down to the control.

Parameters

param1

pqmsg *(PQMSG)*
Message queue structure.

This points to a QMSG structure.

param2 *(BIT32)*
Reserved.

NULL Reserved value.

Returns

reply

dialogcode *(ULONG)*
Dialog code information flags.

These identify the type of control:

DLGC_ENTRYFIELD	Identifies an entry field control. Assumed to understand the EM_SETSEL message.
DLGC_BUTTON	Identifies a button item. Assumed to understand the BM_CLICK message.
DLGC_CHECKBOX	Identifies a checkbox item. Used with the DLGC_BUTTON code.
DLGC_RADIOBUTTON	Identifies a radio button control. Used with the DLGC_BUTTON code.
DLGC_STATIC	Identifies a static control. Static controls are not included in arrow key enumeration.

DLGC_DEFAULT	Identifies a default pushbutton control.
DLGC_PUSHBUTTON	Identifies a nondefault pushbutton.
DLGC_SCROLLBAR	Identifies a scroll bar control.
DLGC_MENU	Identifies a menu control.
DLGC_MLE	Identifies a multi-line entry field control.

Remarks

When processing user input, the dialog manager makes some assumptions about the operation of specific controls. The dialog manager sends the WM_QUERYDLGCODE message to obtain a code that governs what assumptions can be made.

If the window receiving this message is *not* a control as defined above, this message returns 0.

Default Processing

The default dialog procedure takes no action on this message, other than to set **dialogcode** to NULL.

Language Support Window Processing

This system-provided window procedure processes messages for a window that has been created with a window class specifying a 'NULL' window procedure.

General Window Messages

The following describes the WM_* messages and the language support window procedure action.

WM_ACTIVATE

For the cause of this message, see "WM_ACTIVATE" on page 12-3.

Parameters

For a description of the parameters, see "WM_ACTIVATE" on page 12-3.

Remarks

The Language Support Window Procedure responds to this message by posting a WM_PACTIVATE message to the application queue and setting **reply** to NULL.

Default Processing

The default window procedure takes no action on this message, other than to set **reply** to NULL.

WM_CONTROL

For the cause of this message, see "WM_CONTROL" on page 12-17.

Parameters

For a description of the parameters, see "WM_CONTROL" on page 12-17.

Remarks

The Language Support Window Procedure responds to this message by posting a WM_PCONTROL message to the application queue and setting **reply** to NULL.

Default Processing

The default window procedure takes no action on this message, other than to set **reply** to NULL.

WM_PAINT

For the cause of this message, see "WM_PAINT" on page 12-30.

Parameters

For a description of the parameters, see "WM_PAINT" on page 12-30.

Remarks

The Language Support Window Procedure responds to this message by posting a WM_PPAINT message to the application queue and setting **reply** to NULL.

The WinBeginPaint and WinEndPaint calls are issued by the Language Support Window Procedure, during the processing of the WM_PPAINT message.

Default Processing

The default window procedure issues the WinBeginPaint and WinEndPaint calls, and then sets **reply** to NULL.

WM_PPAINT

For the cause of this message, see "WM_PPAINT" on page 12-30.

Parameters

For a description of the parameters, see "WM_PPAINT" on page 12-30.

Remarks

The Language Support Window Procedure issues the WinBeginPaint and WinEndPaint calls, and then sets **reply** to NULL.

Default Processing

The default window procedure takes no action on this message, other than to set **reply** to NULL.

WM_SETFOCUS

For the cause of this message, see "WM_SETFOCUS" on page 12-39.

Parameters

For a description of the parameters, see "WM_SETFOCUS" on page 12-39.

Remarks

The Language Support Window Procedure responds to this message by posting a WM_PSETFOCUS message to the application queue and setting **reply** to NULL.

Default Processing

The default window procedure takes no action on this message, other than to set **reply** to NULL.

WM_SIZE

For the cause of this message, see "WM_SIZE" on page 12-42.

Parameters

For a description of the parameters, see "WM_SIZE" on page 12-42.

Remarks

The Language Support Window Procedure responds to this message by posting a WM_PSIZE message to the application queue and setting **reply** to NULL.

Default Processing

The default window procedure takes no action on this message, other than to set **reply** to NULL.

WM_SYSCOLORCHANGE

For the cause of this message, see "WM_SYSCOLORCHANGE" on page 12-43.

Parameters

For a description of the parameters, see "WM_SYSCOLORCHANGE" on page 12-43.

Remarks

The Language Support Window Procedure responds to this message by posting a WM_PSYSCOLORCHANGE message to the application queue and setting **reply** to NULL.

Default Processing

The default window procedure takes no action on this message, other than to set **reply** to NULL.

Language Support Dialog Processing

This system-provided window procedure processes messages for a dialog that has been created or loaded specifying a 'NULL' dialog procedure.

For any other messages the Language Support Dialog Procedure issues and returns the result of the WinDefDlgProc call.

WM_ACTIVATE

For the cause of this message, see "WM_ACTIVATE" on page 12-3.

Parameters

For a description of the parameters, see "WM_ACTIVATE" on page 12-3.

Remarks

The Language Support Dialog Procedure responds to this message by issuing the WinDefDlgProc call, then posting a WM_PACTIVATE message to the application queue and setting **reply** to the result of the WinDefDlgProc call.

Default Processing

The default window procedure takes no action on this message, other than to set **reply** to NULL.

WM_CONTROL

For the cause of this message, see "WM_CONTROL" on page 12-17.

Parameters

For a description of the parameters, see "WM_CONTROL" on page 12-17.

Remarks

The Language Support Dialog Procedure responds to this message by issuing the WinDefDlgProc call, then posting a WM_PCONTROL message to the application queue and setting **reply** to the result of the WinDefDlgProc call.

Default Processing

The default window procedure takes no action on this message, other than to set **reply** to NULL.

WM_PAINT

For the cause of this message, see "WM_PAINT" on page 12-30.

Parameters

For a description of the parameters, see "WM_PAINT" on page 12-30.

Remarks

The Language Support Dialog Procedure responds to this message by issuing the WinDefDlgProc call, then posting a WM_PPAINT message to the application queue and setting **reply** to the result of the WinDefDlgProc call.

The WinBeginPaint and WinEndPaint calls are issued by the Language Support Dialog Procedure, during the processing of the WM_PPAINT message.

Default Processing

The default window procedure issues the WinBeginPaint and WinEndPaint calls, and then sets **reply** to NULL.

WM_PPAINT

For the cause of this message, see "WM_PPAINT" on page 12-30.

Parameters

For a description of the parameters, see "WM_PPAINT" on page 12-30.

Remarks

The Language Support Dialog Procedure issuing the WinDefDlgProc call, then issues the WinBeginPaint and WinEndPaint calls, and then setting **reply** to the result of the WinDefDlgProc call.

Default Processing

The default window procedure takes no action on this message, other than to set **reply** to NULL.

WM_SETFOCUS

For the cause of this message, see "WM_SETFOCUS" on page 12-39.

Parameters

For a description of the parameters, see "WM_SETFOCUS" on page 12-39.

Remarks

The Language Support Dialog Procedure responds to this message by issuing the WinDefDlgProc call, then posting a WM_PSETFOCUS message to the application queue and setting **reply** to the result of the WinDefDlgProc call.

Default Processing

The default window procedure takes no action on this message, other than to set **reply** to NULL.

WM_SIZE

For the cause of this message, see "WM_SIZE" on page 12-42.

Parameters

For a description of the parameters, see "WM_SIZE" on page 12-42.

Remarks

The Language Support Dialog Procedure responds to this message by issuing the WinDefDlgProc call, then posting a WM_PSIZE message to the application queue and setting **reply** to the result of the WinDefDlgProc call.

Default Processing

The default window procedure takes no action on this message, other than to set **reply** to NULL.

WM_SYSCOLORCHANGE

For the cause of this message, see "WM_SYSCOLORCHANGE" on page 12-43.

Parameters

For a description of the parameters, see "WM_SYSCOLORCHANGE" on page 12-43.

Remarks

The Language Support Dialog Procedure responds to this message by issuing the WinDefDlgProc call, then posting a WM_PSYSCOLORCHANGE message to the application queue and setting **reply** to the result of the WinDefDlgProc call.

Default Processing

The default window procedure takes no action on this message, other than to set **reply** to NULL.

Chapter 13. Button Control Window Processing

This system-provided window procedure processes the actions on a button control (WC_BUTTON).

Button Control Styles

Button controls have these basic window styles:

Button Control Style	Meaning
BS_PUSHBUTTON	A pushbutton is a box that contains a string. When a button is pushed, by clicking the pointing device on it or pressing the spacebar when it is active, the parent window is notified.
BS_CHECKBOX	A checkbox is a small square with a character string to the right. If it is checked, a small black box appears inside the small square. When the box or string is clicked, by clicking on it with the pointing device or pressing the keyboard spacebar when it is active, the checkbox changes state and the parent window is notified.
BS_AUTOCHECKBOX	An automatic check box automatically toggles its state whenever the user clicks on it.
BS_RADIOBUTTON	A radio button is similar to a check box, but is typically used in groups in which only one button at a time is checked. When a radio button is clicked or a cursor key is pressed to move within the group, it notifies its owner window. It is then up to the owner window to check the clicked radio button and uncheck all the rest, if necessary.
BS_AUTORADIOBUTTON	When clicked, an automatic radio button automatically checks itself and unchecks all other radio buttons in the same group.
BS_3STATE	A three-state check box is identical to a check box control except that its check box can be halftoned as well as the box being checked or unchecked.
BS_AUTO3STATE	An automatic three-state check box automatically toggles its state when the user clicks on it.
BS_USERBUTTON	This is an application-definable button. The owner window of this style control receives the additional button style BN_PAINT.

This style can be ORed with any of the basic button styles:

BS_NOPOINTERFOCUS	Buttons with this style do not set the focus to themselves when clicked with the pointing device. This enables the cursor to stay on a control for which information is required, rather than moving to the button.

These styles can be ORed with the BS_PUSHBUTTON style:

BS_HELP	The button posts a WM_HELP message rather than a WM_COMMAND message.
BS_SYSCOMMAND	The button posts a WM_SYSCOMMAND message rather than a WM_COMMAND message.
BS_NOBORDER	The pushbutton is displayed without a border drawn around it. There is no other change in the pushbutton's operation.

If both BS_HELP and BS_SYSCOMMAND are set, BS_HELP takes precedence.

This style can be ORed with the BS_PUSHBUTTON and BS_USERBUTTON styles to give the BS_DEFPUSHBUTTON and BS_DEFUSERBUTTON styles:

BS_DEFAULT A BS_DEFAULT pushbutton is one with a thick border box. It has the same properties as a pushbutton. In addition, the user may press a BS_DEFAULT pushbutton by pressing the RETURN or ENTER key. The intention is the same for userbuttons, but the appearance of a BS_DEFAULT userbutton is application-defined.

Button Control Data

BTNCDATA Button-control-data structure.

length *(COUNT2B)*
Length of the control data in bytes.

8 The length of the control data for a button control.

checkstate *(BIT16)*
Check state of button.

This is the same value as returned by the BM_QUERYCHECK message and passed to the BM_SETCHECK message.

hilite *(BIT16)*
Highlighting state of button.

This is the same value as returned by the BM_QUERYHILITE message and passed to the BM_SETHILITE message.

Button Control Notification Messages

These messages are initiated by the button control window to notify its owner of significant events.

WM_COMMAND

For the cause of this message, see "WM_COMMAND" on page 12-16.

Parameters

For a description of the parameters, see "WM_COMMAND" on page 12-16.

Button control sets **cmd** to the button identity and **source** to CMDSRC_PUSHBUTTON.

Remarks

The button control generates this message and posts it to the queue of its owner, if its style is BS_PUSHBUTTON or BS_DEFPUSHBUTTON, when a pushbutton is pressed, or when it receives a BM_CLICK message.

Default Processing

The default window procedure takes no action on this message, other than to set **reply** to NULL.

WM_CONTROL

For the cause of this message, see "WM_CONTROL" on page 12-17.

Parameters

param1

id *(IDENTITY)*
Button control identity.

notifycode *(USHORT)*
Notification code.

The notification code BN_PAINT is only generated when the button control has a style of BS_USERBUTTON.

The button control uses these notification codes:

BN_CLICKED	The button has been pressed.
BN_DBLCLICKED	The button has been double-clicked.
BN_PAINT	The button requires painting, using one of the following draw states:
	BDS_DISABLED The disabled state of the button requires painting.
	BDS_HILITED The highlighted state of the button requires painting.
	BDS_DEFAULT The default state of the button requires painting.

param2

controlspec *(BIT32)*
Control-specific information.

When **notifycode** is BN_PAINT this parameter is a pointer to a *USERBUTTON* structure, otherwise this parameter is the window handle of the button control.

Returns

reply *(BIT32)*
Reserved.

NULL Reserved value.

Remarks

The button control generates this message and sends it to its owner, informing the owner of this event, when:

- Its style is BS_CHECKBOX, BS_RADIOBUTTON, or BS_3STATE, and the button is pressed.
- It receives a BM_CLICK message.
- Its style is BS_USERBUTTON and the button is clicked or double clicked.

Default Processing

The default window procedure takes no action on this message, other than to set **reply** to NULL.

WM_HELP

For the cause of this message, see "WM_HELP" on page 12-22.

Parameters

For a description of the parameters, see "WM_HELP" on page 12-22.

Button control sets **cmd** to the button identity.

Remarks

This message is identical to a WM_COMMAND message, but implies that the application should respond to this message by displaying help information.

The button control generates this message and posts it to the queue of its owner, if it has the style of BS_HELP and a pushbutton is pressed, or when it receives a BM_CLICK message.

Default Processing

The default window procedure sends this message to the parent window, if it exists and is not the desktop. Otherwise, it sets **reply** to NULL.

WM_SYSCOMMAND

For the cause of this message, see "WM_SYSCOMMAND" on page 12-43.

Parameters

For a description of the parameters, see "WM_SYSCOMMAND" on page 12-43.

Button control sets **cmd** to the button identity.

Remarks

If the button control is specified with a style of BS_SYSCOMMAND but not with BS_HELP, the button control generates this message and posts it to the queue of its owner when a pushbutton is pressed, or when it receives a BM_CLICK message.

Default Processing

The default window procedure takes no action on this message, other than to set **reply** to NULL.

Button Control Window Messages

This section describes the Button Control Window Procedure actions on receiving the following messages.

BM_CLICK

An application sends this message to cause the effect of the operator clicking a pushbutton.

Parameters

param1

up *(BOOL)*
Up/down indicator:

TRUE Perform the default upclick action
FALSE Perform the default downclick action.

param2 *(BIT32)*
Reserved.

NULL Reserved value.

Returns

reply *(BIT32)*
Reserved.

NULL Reserved value.

Remarks

The button control responds to this message by taking the action that occurs if the button is clicked by the operator. This causes the following messages to be generated:

- A WM_HELP message, if the button has a style of BS_HELP.
- A WM_SYSCOMMAND message, if the button has a style of BS_PUSHBUTTON and a style of BS_SYSCOMMAND and not a style of BS_HELP.
- A WM_COMMAND message, if the button has a style of BS_PUSHBUTTON but not a style of BS_SYSCOMMAND and not a style of BS_HELP.
- A WM_CONTROL message, whose **notifycode** is set to BN_CLICKED, if the button has a style of BS_USERBUTTON, BS_PUSHBUTTON, BS_CHECKBOX, or BS_3STATE, and not a style of BS_SYSCOMMAND or BS_HELP.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set **reply** to the default value of NULL.

BM_QUERYCHECK

This message returns the checked state of a button control.

Parameters

param1 *(BIT32)*
Reserved.

NULL Reserved value.

param2 *(BIT32)*
Reserved.

NULL Reserved value.

Returns

reply

result *(USHORT)*
Check indicator:

0 The button control is in unchecked state
1 The button control is in checked state
2 The button control is in indeterminate state.

Remarks

The button control responds to this message, if it has a style of BS_CHECKBOX, BS_AUTOCHECKBOX, BS_RADIOBUTTON, BS_AUTORADIOBUTTON, BS_3STATE, or BS_AUTO3STATE, by setting **result** as appropriate.

If the button has any other style, the button control takes no action other than to set **result** to 0.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set **result** to the default value of NULL, which is equivalent to zero.

BM_QUERYCHECKINDEX

This message returns the zero-based index of a checked radio button.

Parameters

param1 *(BIT32)*
Reserved.

NULL Reserved value.

param2 *(BIT32)*
Reserved.

NULL Reserved value.

Returns

reply

result *(SHORT)*
Radio-button index:

–1 No radio button of the group is checked, or this button control does not have the style BS_RADIOBUTTON or BS_AUTORADIOBUTTON.
Other Zero-based index of the checked radio button of the group.

Remarks

The button control responds to this message by setting **result** as appropriate.

This message may be sent to any radiobutton or autoradiobutton in a group of buttons. For details of the WS_GROUP style, see Chapter 12, "Default Window Procedure Message Processing."

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set **result** to the default value of NULL, which is equivalent to zero.

BM_QUERYHILITE

This message returns the highlighting state of a button control.

Parameters

param1 *(BIT32)*
Reserved.

NULL Reserved value.

param2 *(BIT32)*
Reserved.

NULL Reserved value.

Returns

reply

result *(BOOL)*
Highlight indicator:

TRUE The button control is displayed in highlighted state
FALSE The button control is displayed in unhighlighted state.

Remarks

The button control responds to this message, if it has a style of BS_PUSHBUTTON, by setting **result** as appropriate.

If the button has any other style, the button control takes no action other than to set **result** to FALSE.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, except to set **result** to the default value of NULL, which is equivalent to FALSE.

BM_SETCHECK

This message sets the checked state of a button control.

Parameters

param1

check *(USHORT)*
Check state:

0 Display the button control in the unchecked state
1 Display the button control in the checked state
2 Display a 3-state button control in the indeterminate state.

param2 *(BIT32)*
Reserved.

NULL Reserved value.

Returns

reply

oldstate *(USHORT)*
Old check state of the button control:

0 Unchecked
1 Checked
2 Indeterminate.

Remarks

The button control responds to this message by displaying it in the appropriate state and returning the old state.

If the button control has the style of BS_CHECKBOX, BS_AUTOCHECKBOX, BS_RADIOBUTTON, or BS_AUTORADIOBUTTON, it is displayed in the checked state if **check** is set to 1, or in the unchecked state if it is set to 0 and **oldstate** is set as appropriate.

If the button control has the style of BS_RADIOBUTTON or BS_AUTORADIOBUTTON, the WS_TABSTOP style is modified. If the resulting state of the button is checked, the WS_TABSTOP style is set, otherwise it is reset.

If the button control has the style of BS_3STATE or BS_AUTO3STATE, it is displayed in the unchecked state if **check** is set to 0, in the checked state if it is set to 1, and in the indeterminate state if it is set to 2 and **oldstate** is set as appropriate.

If the button control has the style of BS_USERBUTTON, a WM_CONTROL message is sent to its owner with **notifycode** set to BN_PAINT and **oldstate** is set as appropriate.

If the button control has any other style, the button control takes no action other than to set **oldstate** to 0.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, except to set **oldstate** to the default value of NULL, which is equivalent to zero.

BM_SETDEFAULT

This message sets the default state of a button control.

Parameters

param1

default *(BOOL)*
Default state:

TRUE Display the button control in the default state
FALSE Display the button control in the nondefault state.

param2 *(BIT32)*
Reserved.

NULL Reserved value.

Returns

reply

success *(BOOL)*
Success indicator:

TRUE Successful operation
FALSE Error occurred.

Remarks

The button control responds to this message, if it has a style of BS_USERBUTTON or BS_PUSHBUTTON, by displaying the button control in the default or nondefault state as appropriate, and setting **success** to TRUE.

If the button control has any other style, the button control takes no action other than to set **success** to TRUE.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set **success** to the default value of NULL, which is equivalent to FALSE.

BM_SETHILITE

This message sets the highlight state of a button control.

Parameters

param1

hilite *(BOOL)*
Highlight indicator:

TRUE	Display the button control in the highlighted state
FALSE	Display the button control in the unhighlighted state.

param2 *(BIT32)*
Reserved.

NULL	Reserved value.

Returns

reply

oldstate *(BOOL)*
Old highlight state:

TRUE	The button control was in highlighted state
FALSE	The button control was in unhighlighted state.

Remarks

The button control responds to this message, if it has a style of BS_PUSHBUTTON, BS_CHECKBOX, BS_AUTOCHECKBOX, BS_RADIOBUTTON, BS_AUTORADIOBUTTON, BS_3STATE, or BS_AUTO3STATE, by displaying the button control in the appropriate highlight state and setting **oldstate** as appropriate.

If the style of the Button Control is BS_USERBUTTON, a WM_CONTROL message is sent to its owner with **notifycode** set to BN_PAINT and with **controlspec** pointing to a *USERBUTTON* structure and sets **oldstate** as appropriate.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set **oldstate** to the default value of NULL, which is equivalent to FALSE.

WM_ENABLE

For the cause of this message, see "WM_ENABLE" on page 12-20.

Parameters

For a description of the parameters, see "WM_ENABLE" on page 12-20.

Remarks

The button control window procedure responds to this message by setting the enable state and by setting **reply** to NULL.

Default Processing

The default window procedure takes no action on this message, other than to set **reply** to NULL.

WM_MATCHMNEMONIC

For the cause of this message, see "WM_MATCHMNEMONIC" on page 12-52.

Parameters

For a description of the parameters, see "WM_MATCHMNEMONIC" on page 12-52.

Remarks

The button control window procedure responds to this message by setting **result** as appropriate.

Default Processing

The default window procedure takes no action on this message, other than to set **result** to FALSE.

WM_QUERYCONVERTPOS

For the cause of this message, see "WM_QUERYCONVERTPOS" on page 12-33.

Parameters

For a description of the parameters, see "WM_QUERYCONVERTPOS" on page 12-33.

Remarks

The button control window procedure returns QCP_NOCONVERT.

Default Processing

For the default window procedure processing of this message see "WM_QUERYCONVERTPOS" on page 12-33.

WM_QUERYWINDOWPARAMS

Occurs when an application queries the button control window procedure window parameters.

Parameters

For a description of the parameters, see "WM_QUERYWINDOWPARAMS" on page 12-35.

Remarks

The button control window procedure responds to this message by passing it to the default window procedure.

Default Processing

The default window procedure sets the **length**, **presparamslength**, and **ctldatalength** parameters of the *WNDPARAMS* data structure, identified by **wndparams**, to zero and sets **result** to FALSE.

WM_SETWINDOWPARAMS

Occurs when an application sets or changes the button control window procedure window parameters.

Parameters

For a description of the parameters, see "WM_SETWINDOWPARAMS" on page 12-40.

Remarks

The button control window procedure responds to this message by passing it to the default window procedure.

Default Processing

The default window procedure takes no action on this message, other than to set **result** to FALSE.

Chapter 14. Entry Field Control Window Processing

This system-provided window procedure processes the actions on an entry field control (WC_ENTRYFIELD).

Purpose

An entry field control is a rectangular window that displays a single line of text that the operator can edit. When it has the focus, the cursor marks the current **insertion** or **replacement** point.

Entry Field Control Styles

These entry field control styles are available:

ES_LEFT The text in the control is left-justified. This is the default style if neither ES_RIGHT nor ES_CENTER is specified.

ES_RIGHT The text in the control is right-justified.

ES_CENTER The text in the control is centered.

ES_MARGIN This style can be used to cause a frame to be drawn around the control, with a margin around the editable text. The margin is half a character-width wide and half a character-height high.

When an entry field control with this style is positioned, it adjusts the position so that the text is placed at the position specified. This position differs from the original position by the width of the border and the margin.

ES_READONLY This style causes a single line entry field to be created in read only state.

When an entry field is in read only state, characters do not get inserted into the text. However the insertion interface is still functional.

The entry field read only state can be altered by use of the EM_SETREADONLY message.

ES_UNREADABLE This style causes the text not to be displayed (for passwords, for example).

These entry field controls are intended for countries that use a double-byte character encoding scheme:

ES_SBCS The text is purely single-byte.

If the number of characters entered exceeds EM_SETTEXTLIMIT, or a DBCS character is entered, the alarm sounds and the last character entered is ignored.

ES_DBCS The text is purely double byte.

If the number of bytes in the entry field exceeds EM_SETTEXTLIMIT, or an SBCS character is entered, the alarm sounds and the last character entered is ignored.

ES_ANY The text is a mixture of SBCS and DBCS characters.

If the number of bytes in the input field exceeds EM_SETTEXTLIMIT, the alarm sounds and the last character entered is ignored.

ES_ANY is the default.

Note: If the queue code page is an ASCII code page and the data in the entry field is to be converted to an EBCDIC code page, there is a possibility that shift-in and shift-out characters introduced by the conversion process can cause the converted data to overrun the target field. Coding ES_MIXED protects the target field from overrun in this situation.

ES_MIXED The text is a mixture of SBCS and DBCS characters which may subsequently be converted from an ASCII DBCS code page to an EBCDIC DBCS code page with a consequent possible increase in the length of the data.

If

```
DBCSchars*2 + SBCSchars + N > EM_SETTEXTLIMIT
```

where N starts at 0 and is incremented whenever the string goes from SBCS to DBCS or DBCS to SBCS, the alarm sounds and the last character entered is ignored.

Note: For every conversion from SBCS to DBCS there must be a corresponding return to SBCS (N must be an even number).

Entry Field Control Data

ENTRYFDATA Entry-field control data structure.

length *(COUNT2B)*
Length of control data in bytes.

8 The length of the control data for an entry field control.

editlimit *(COUNT2CH)*
Edit limit.

This is the maximum number of characters that can be entered into the entry field control.

If the operator tries to enter more text into an entry field control than is specified by the text limit set by the EM_SETTEXTLIMIT message, the entry field control indicates the error by sounding the alarm and does not accept the characters.

minsel *(USHORT)*
Minimum selection.

maxsel *(USHORT)*
Maximum selection.

The **minsel** and **maxsel** parameters identify the current selection within the entry field control. Characters within the text with byte offsets less than the **maxsel** parameter and greater than or equal to the **minsel** parameter are the current selection. The cursor is positioned immediately before the character identified by the **maxsel** parameter.

If the **minsel** parameter is equal to the **maxsel** parameter, the current selection becomes the insertion point.

If the **minsel** parameter is equal to zero and the **maxsel** is greater than or equal to text limit set by the EM_SETTEXTLIMIT message, the entire text is selected.

Entry Field Control Notification Messages

This message is initiated by the entry field control window to notify its owner of significant events.

WM_CONTROL

For the cause of this message, see "WM_CONTROL" on page 12-17.

Parameters

param1

id *(USHORT)*
Control window identity.

notifycode *(USHORT)*
Notify code:

EN_CHANGE The content of the entry field control has changed, and the change has been displayed on the screen.

EN_KILLFOCUS The entry field control is losing the focus.

EN_MEMERROR The entry field control cannot allocate the storage necessary to accommodate window text of the length implied by the EM_SETTEXTLIMIT message.

EN_OVERFLOW The entry field control cannot insert more text than the current text limit.

If the recipient of this message returns TRUE, then the entry field control retries the operation, otherwise it terminates the operation.

EN_SCROLL The entry field control is about to scroll horizontally. This can happen in these circumstances:

- The application has issued a WinScrollWindow call
- The content of the entry field control has changed
- The caret has moved.

The entry field control must scroll to show the caret position.

EN_SETFOCUS The entry field control is receiving the focus.

param2

controlspec *(HWND)*
Entry field control window handle.

Returns

reply *(BIT32)*
Reserved.

NULL Reserved value.

Remarks

The entry field control window procedure generates this message and sends it to its owner, informing the owner of the event.

Default Processing

The default window procedure takes no action on this message, other than to set **reply** to NULL.

Entry Field Control Window Messages

This section describes the entry field control window procedure actions on receiving these messages:

EM_CLEAR

This message deletes the text that forms the current selection.

Parameters

param1 *(BIT32)*
Reserved.

NULL Reserved value.

param2 *(BIT32)*
Reserved.

NULL Reserved value.

Returns

reply

success *(BOOL)*
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Remarks

The entry field control window procedure responds to this message by deleting the text that forms the current selection and setting **maxsel** equal to **minsel**.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set **success** to the default value of NULL, which is equivalent to FALSE.

EM_COPY

This message sends the current selection to the clipboard.

Parameters

param1 *(BIT32)*
Reserved.

NULL Reserved value.

param2 *(BIT32)*
Reserved.

NULL Reserved value.

Returns

reply

success *(BOOL)*
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Remarks

The entry field control window procedure responds to this message by sending the text that forms the current selection to the clipboard in CF_TEXT format.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set **success** to the default value of NULL, which is equivalent to FALSE.

EM_CUT

This message sends the text that forms the current selection to the clipboard, and then deletes it from the entry field control.

Parameters

param1 *(BIT32)*
Reserved.

NULL Reserved value.

param2 *(BIT32)*
Reserved.

NULL Reserved value.

Returns

reply

success *(BOOL)*
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Remarks

The entry field control window procedure responds to this message by sending the text that forms the current selection to the clipboard in CF_TEXT format, and then deleting it from the entry field control and setting **maxsel** equal to **minsel**.

This message is the combination of a EM_COPY message followed by a EM_CLEAR message.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set **success** to the default value of NULL, which is equivalent to FALSE.

EM_PASTE

This message replaces the text that forms the current selection with text from the clipboard.

Parameters

param1 *(BIT32)*
Reserved.

NULL Reserved value.

param2 *(BIT32)*
Reserved.

NULL Reserved value.

Returns

reply

success *(BOOL)*
Success indicator:

TRUE Successful completion
FALSE Error occurred.

For example, if the text to be inserted does not fit in the entry field control without overflowing the text limit set by the EM_SETTEXTLIMIT message, in which instance no text is inserted.

Remarks

The entry field control window procedure responds to this message by replacing the text that forms the current selection with text from the clipboard, if the data is in CF_TEXT format.

Only characters from the clipboard up to the first carriage return are used in the replacement.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set **success** to the default value of NULL, which is equivalent to FALSE.

EM_QUERYCHANGED

This message enquires if the text of the entry field control has been changed since the last enquiry.

Parameters

param1 *(BIT32)*
Reserved.

NULL Reserved value.

param2 *(BIT32)*
Reserved.

NULL Reserved value.

Returns

reply

changed *(BOOL)*
Changed indicator:

TRUE The text in the entry field control has been changed since the last time it received this message or a WM_QUERYWINDOWPARAMS message.
FALSE All other situations.

Remarks

The entry field control window procedure responds to this message by setting **changed** to indicate whether the text of the entry field has been changed since the last time either this message or a WM_QUERYWINDOWPARAMS message has been received.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set **changed** to the default value of NULL, which is equivalent to FALSE.

EM_QUERYFIRSTCHAR

This message returns the zero-based offset of the first character displayed in the entry field control.

Parameters

param1 *(BIT32)*
Reserved.

NULL Reserved value.

param2 *(BIT32)*
Reserved.

NULL Reserved value.

Returns

reply

offset *(SHORT)*
Zero-based offset of the first character displayed.

Remarks

The entry field control window procedure responds to this message by returning the zero-based offset into the text that corresponds to the first character displayed in the entry field control.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set **offset** to the default value of NULL, which is equivalent to zero.

EM_QUERYSEL

This message gets the zero-based offsets of the bounds of the text that forms the current selection.

Parameters

param1 *(BIT32)*
Reserved.

NULL Reserved value.

param2 *(BIT32)*
Reserved.

NULL Reserved value.

Returns

reply

minsel *(SHORT)*
Offset of the first character in the selection.

maxsel *(SHORT)*
Offset of the first character after the selection.

Remarks

The entry field control window procedure responds to this message by returning the zero-based offsets of the bounds of the text that forms the current selection.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set **reply** to the default value of NULL, which is equivalent to setting both **minsel** and **maxsel** to zero.

EM_SETFIRSTCHAR

This message specifies the offset of the character to be displayed in the first position of the entry field control.

Parameters

param1

offset *(SHORT)*
Zero-based offset of the first character to be displayed.

param2 *(BIT32)*
Reserved.

NULL Reserved value.

Returns

reply

success *(BOOL)*
Success indicator:

TRUE Successful completion
FALSE Error occurred. For example, because **offset** is not valid.

Remarks

The entry field control window procedure responds to this message by setting the text displayed in the edit control so that the first character displayed on the left of the window has the zero-based index specified by **offset**.

An EN_SCROLL notification message occurs, if the entry field control scrolls.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set **success** to the default value of NULL, which is equivalent to FALSE.

EM_SETREADONLY

This message sets the read only state of an entry field control.

Parameters

param1

readonly *(BOOL)*
Read only state indicator:

TRUE Enable read only state
FALSE Disable read only state.

param2 *(BIT32)*
Reserved.

NULL Reserved value.

Returns

reply

oldreadonly *(BOOL)*
Previous read only state indicator:

TRUE Read only state was previously enabled
FALSE Read only state was previously disabled.

Remarks

The entry field control window procedure responds to this message by setting the read only state of the entry field control.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set **oldreadonly** to the default value of NULL, which is equivalent to FALSE.

EM_SETSEL

This message sets the zero-based offsets of the bounds of the text that forms the current selection.

Parameters

param1

minsel *(USHORT)*
Offset of the first character in the selection.

maxsel *(USHORT)*
Offset of the first character after the selection.

If **minsel** equals **maxsel**, the current selection becomes an insertion point.

If **minsel** equals zero and **maxsel** is equal to or greater than the text limit set by the EM_SETTEXTLIMIT message, the entire text is selected. Selected text is displayed in reverse color.

param2 *(BIT32)*
Reserved.

NULL Reserved value.

Returns

reply

success *(BOOL)*
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Remarks

The entry field control window procedure responds to this message by setting the zero-based offsets of the bounds of the text that forms the current selection.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set **success** to the default value of NULL, which is equivalent to FALSE.

EM_SETTEXTLIMIT

This message sets the maximum number of bytes that an entry field control can contain.

Parameters

param1

textlimit *(SHORT)*
Maximum number of characters in the entry field control.

param2 *(BIT32)*
Reserved.

NULL Reserved value.

Returns

reply

success *(BOOL)*
Success indicator:

TRUE	Successful completion
FALSE	Error occurred. For example, because not enough storage can be allocated.

Remarks

The entry field control window procedure responds to this message by setting the maximum number of characters that can be contained.

This message is intended only to limit the length of lines that result from the user interacting with the entry field control. It also limits the length of text that can result from sending a EM_PASTE or WM_SETWINDOWPARAMS message.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set **success** to the default value of NULL, which is equivalent to FALSE.

WM_CHAR

For the cause of this message, see "WM_CHAR" on page 12-14.

Parameters

For a description of the parameters, see "WM_CHAR" on page 12-14.

Remarks

The entry field control window procedure responds to this message by sending it to its owner if it has not processed the key stroke. This is the most common means by which the input focus is switched around the various controls in a dialog box.

Unlike other controls, the **vk** field of the message "WM_CHAR" takes precedence over other fields only when the shift key is pressed.

If this message contains a valid **ch** field of the message "WM_CHAR," that character is entered into the text in insert or overtype mode.

The keystrokes processed by an entry field control are:

Left arrow	Move the cursor one character to the left.
Right arrow	Move the cursor one character to the right.
Shift + Left arrow	Extend the selection by one character to the left.
Shift + Right arrow	Extend the selection by one character to the right.
Home	Move the cursor to the beginning of the text.
End	Move the cursor to the end of the text.
Backspace	Delete the character to the left of the cursor.
Delete	When the selection is an insertion point, delete the character to the right of the cursor, otherwise delete the current selection, but do not put it in the clipboard.
Shift + Del	Cut the current selection to the clipboard.
Shift + Ins	Replace the current selection with the text contents from the clipboard.
Ctrl + Del	Delete to the end of the field.
Ctrl + Ins	Copy the current selection to the clipboard.

If the control contains more text than can be shown, the actions defined above that move the cursor cause the text to be scrolled. The amount of scrolling varies from key to key, and the position of the text within the control varies for the same cursor position.

Default Processing

The default window procedure sends the message to the owner window if it exists, otherwise it takes no action on this message other than to set **result** to FALSE.

WM_QUERYCONVERTPOS

For the cause of this message, see "WM_QUERYCONVERTPOS" on page 12-33.

Parameters

For a description of the parameters, see "WM_QUERYCONVERTPOS" on page 12-33.

Remarks

The entry field control window procedure returns QCP_CONVERT, and updates **cursorpos** to the position of the cursor.

Default Processing

For the default window procedure processing of this message see "WM_QUERYCONVERTPOS" on page 12-33.

WM_QUERYWINDOWPARAMS

This message occurs when an application queries the entry field control window parameters.

Parameters

For a description of the parameters, see "WM_QUERYWINDOWPARAMS" on page 12-35.

Remarks

The entry field control window procedure responds to this message by returning the window parameters indicated by the **status** parameter of the *WNDPARAMS* data structure, identified by the **wndparams** parameter.

Default Processing

The default window procedure sets the **length**, **presparamslength**, and **ctldatalength** parameters of the *WNDPARAMS* data structure, identified by **wndparams**, to zero and sets **result** to FALSE.

WM_SETWINDOWPARAMS

This message occurs when an application sets or changes the entry field control window parameters.

Parameters

For a description of the parameters, see "WM_SETWINDOWPARAMS" on page 12-40.

Remarks

The entry field control window procedure responds to this message by setting the window parameters indicated by the **status** parameter of the *WNDPARAMS* data structure, identified by the **wndparams** parameter.

Default Processing

The default window procedure takes no action on this message, other than to set **result** to FALSE.

Chapter 15. Frame Control Window Processing

This system-provided window procedure processes the actions on a frame window (WC_FRAME). The frame control window procedure sends all messages not processed to FID_CLIENT and sets **reply** to NULL.

Purpose

The window that contains all of the parts listed below is called the *frame window*. Each of the parts that make up a window, such as the title bar and menu, are separate child windows of the frame window. All of these child windows, except the client window (FID_CLIENT), are called *frame controls*.

FID_CLIENT is not a frame control, it is an instance of a window class implemented by the application.

The frame window and all of the frame controls are implemented with system-provided preregistered window classes.

The frame window holds together all of the frame controls and FID_CLIENT that make up an application window. The frame window is responsible for arranging the frame controls and the FID_CLIENT as the frame window is sized and moved. It is also responsible for routing specific messages to its frame controls and FID_CLIENT.

How to use

Each of the frame controls and FID_CLIENT are known to the frame window by a system-provided window-identifier value as listed below:

FID_CLIENT	Client window
FID_HORZSCROLL	Horizontal scroll bar
FID_MENU	Application menu
FID_MINMAX	Minimize/Maximize box
FID_SYSMENU	System menu
FID_TITLEBAR	Title bar
FID_VERTSCROLL	Vertical scroll bar.

For correct operation, only one window per frame must be defined with each of the above FID_* values.

As with all controls, when something happens to a frame control (a scroll-bar click or the menu key is pressed, for example), the frame control notifies its owner with window messages. The owner of each of the frame controls is the frame window itself. Messages that are of interest to FID_CLIENT are sent to it by the frame window.

There are two ways to create a system-provided window and its frame controls. The frame window and all of its associated controls (these being determined from the frame window style being used) can be created, together using the WinCreateStdWindow call. Alternatively, the frame window and each control being used can be individually created using the WinCreateWindow call.

It is conventional to add a title to each window that indicates the program that owns the frame window and, if applicable, the file or document that is being processed. This is normally shown in the form:

```
program - document
```

The frame window can be created with the flag FCF_TASKLIST set so as to match this convention. As the program title is added to the front of the title text, a program can satisfy the convention by supplying " – document" as the window title. If the program does not process several documents in this window, a null title satisfies the convention.

Standard Frame Styles and Frame Creation Flags

Frame Creation Flag	Meaning
FCF_TITLEBAR	Title bar.
FCF_SYSMENU	System menu.
FCF_MENU	Application menu.
FCF_MINMAX	Minimize and Maximize buttons.
FCF_MINBUTTON	Minimize button.
FCF_MAXBUTTON	Maximize button.
FCF_VERTSCROLL	Vertical scroll bar.
FCF_HORZSCROLL	Horizontal scroll bar.
FCF_SIZEBORDER	Wide sizing border.
FCF_BORDER	Window is drawn with a thin border.
FCF_DLGBORDER	Window is drawn with a standard dialog border.
FCF_ACCELTABLE	Causes an accelerator table to be loaded, for this frame window, from the resource file identified on the WinCreateStdWindow call.
FCF_ICON	Window is created with an icon associated with it that is used to represent the window when it is minimized. If present, the **Resource** parameter of the WinCreateStdWindow call must be the identity of an icon. This icon is loaded and associated with the window. When the window is minimized, the icon is shown if the screen is capable of showing it. When the window is destroyed, the icon is also destroyed.
FCF_SHELLPOSITION	The window is created with a size and position determined by the shell, rather than explicitly by the application.
FCF_SYSMODAL	The frame window is System Modal.
FCF_NOBYTEALIGN	When this flag is **not** set, the frame window is adjusted so that window operations, such as moving, can be performed in an optimized manner. For example, some displays can move a window more quickly if the movement is by a multiple of eight pels. If this flag is set, such optimizations are not performed and size and position values are honored.
FCF_TASKLIST	When this flag is set, the program title is added to the front of the frame window text. The resulting string is used as the window title and is also entered on the task list. In this context, the program title is the text string used by the Desktop Manager to identify the program, or the text string specified as a parameter in the START command. If neither string has been defined, the filename and extension of the .EXE file are used as the program title.
FCF_NOMOVEWITHOWNER	The window should not be moved when its owner is moved.
FCF_STANDARD	Same as (FCF_TITLEBAR \| FCF_SYSMENU \| FCF_MINBUTTON \| FCF_MAXBUTTON \| FCF_SIZEBORDER \| FCF_ICON \| FCF_MENU \| FCF_ACCELTABLE \| FCF_SHELLPOSITION \| FCF_TASKLIST). This value is assumed if any Frame Window is created with no Control Data.

Frame Window Style	Meaning

Frame styles may only be used when the frame is created from a dialog template.

FS_SCREENALIGN	The coordinates specifying the location of the dialog box are relative to the top left corner of the screen, rather than being relative to the owner window's origin.
FS_MOUSEALIGN	The coordinates specifying the location of the dialog box are relative to the position of the pointing device pointer at the time the window was created. OS/2 Version 1.2 tries to keep the dialog box on the screen, if possible.

Frame Control Data

FRAMECDATA Frame-control-data structure.

length *(COUNT2B)*
Length.

createflags *(BIT32)*
Frame-creation flags.

resources *(RESID)*
Identifier of required resource.

This is supplied in an environment-dependent manner.

resource *(IDENTITY)*
Resource identifier.

Frame Control Notification Messages

These messages are initiated by the frame control window to notify the FID_CLIENT window.

WM_MINMAXFRAME

This message is sent to a frame window that is being minimized, maximized, or restored.

Parameters

param1

swp *(PSWP)*
Set window position structure.

This points to a *SWP* structure. The structure has the appropriate SWP_* indicators set to describe the operation that is occurring to the window.

param2 *(BIT32)*
Reserved.

NULL Reserved value.

Returns

reply

overridedefault *(BOOL)*
Processed indicator:

TRUE	The message has been processed, the default system actions for the operation specified by the *SWP* parameter to the window, are not to be performed.
FALSE	The message has been ignored, the default system actions for the operation specified by the *SWP* parameter to the window, are to be performed.

Remarks

The window words QWS_XRESTORE, QWS_YRESTORE, QWS_CXRESTORE, and QWS_CYRESTORE for **hwnd** are initialized before this message is sent. The window state has not been changed when this message is sent, and so the WinQueryWindowPos call can be used.

This message is sent by default to the FID_CLIENT window.

The system default actions, if FALSE is returned to this message, are based on the operation specified by the *SWP* parameter.

These actions affect the status of the frame window, and the title button windows and system menu windows contained within it, as follows:

- Window is maximized from a minimized state.
 - Title button windows:

 The RESTORE button window is replaced by a MIN button window and the MAX button window is replaced by a RESTORE button window.
 - System menu window:

 The MINIMUM menu entry is enabled and the MAXIMUM menu entry is disabled.
 - Other changes:

 The frame window has the WS_MAXIMIZED style bit set and the WS_MINIMIZED style bit reset. Also the MS_VERTICALFLIP style bit of the system menu window is reset.
- Window is restored from a minimized state.
 - Title button windows:

 The RESTORE button window is replaced by a MIN button window (the MAX button window is unaltered).
 - System menu window:

 The MINIMUM menu entry is enabled, the RESTORE menu entry is disabled and the SIZE menu entry is enabled.
 - Other changes:

 The frame window has the WS_MINIMIZED style bit and the MS_VERTICALFLIP style bit of the system menu window reset.
- Window is minimized from a maximized state.
 - Title button windows:

 The RESTORE button window is replaced by a MAX button window and the MIN button window is replaced by a RESTORE button window.
 - System menu window:

 The MAXIMUM menu entry is enabled and the MINIMUM menu entry is disabled.
 - Other changes:

 The frame window has the WS_MINIMIZED style bit set and the WS_MAXIMIZED style bit reset. Also the MS_VERTICALFLIP style bit of the system menu window is set.
- Window is restored from a maximized state.
 - Title button windows:

 The RESTORE button window is replaced by a MAX button window (the MIN button window is unaltered).
 - System menu window:

 The MAXIMUM menu entry is enabled, the RESTORE menu entry is disabled and the SIZE menu entry is enabled.

- Other changes:

 The frame window has the WS_MAXIMIZED style bit reset.

- Window is minimized from a restored state.
 - Title-button windows:

 The MIN button window is replaced by a RESTORE button window (the MAX button window is unaltered).

 - System menu window:

 The RESTORE menu entry is enabled, the MINIMUM menu entry is disabled and the SIZE menu entry is disabled.

 - Other changes:

 The frame window has the WS_MINIMIZED style bit set, and the MS_VERTICALFLIP style bit of the system menu window is set.

- Window is maximized from a restored state.
 - Title-button windows:

 The MAX button window is replaced with a RESTORE button window (the MIN button window is unaltered).

 - System menu window:

 The RESTORE menu entry is enabled, the MAXIMUM menu entry is disabled and the SIZE menu entry is disabled.

 - Other changes:

 The frame window has the WS_MAXIMIZED style bit set.

Default Processing

The default window procedure takes no action on this message, other than to set **overridedefault** to FALSE.

Frame Control Window Messages

This section describes the frame control window procedure actions on receiving the following messages.

WM_ACTIVATE

For the cause of this message, see "WM_ACTIVATE" on page 12-3.

Parameters

For a description of the parameters, see "WM_ACTIVATE" on page 12-3.

Remarks

The frame control window procedure responds to this message by first sending a TBM_SETHILITE message to the FID_TITLEBAR control, if it exists, to highlight or unhighlight the title bar. If the style is FCF_DLGBORDER, the border is redrawn in either highlighted or unhighlighted state, as necessary. It then sends the WM_ACTIVATE message to the FID_CLIENT window.

On return, if the window is being deactivated and the pointer is set to the frame window or any of its controls, the pointer is destroyed.

Then it sets **reply** to NULL.

Default Processing

The default window procedure takes no action on this message, other than to set **reply** to NULL.

WM_BUTTON1DBLCLK

For the cause of this message, see "WM_BUTTON1DBLCLK" on page 12-6.

Parameters

For a description of the parameters, see "WM_BUTTON1DBLCLK" on page 12-6.

Default Processing

The frame control window procedure responds to this message by issuing the WinSetActiveWindow call and sets **result** to TRUE.

WM_BUTTON2DBLCLK

For the cause of this message, see "WM_BUTTON2DBLCLK" on page 12-8.

Parameters

For a description of the parameters, see "WM_BUTTON1DBLCLK" on page 12-6.

Default Processing

The frame control window procedure processes this message identically to WM_BUTTON1DBLCLK.

WM_BUTTON1DOWN

For the cause of this message, see "WM_BUTTON1DOWN" on page 12-7.

Parameters

For a description of the parameters, see "WM_BUTTON1DOWN" on page 12-7.

Remarks

This message is posted to the application queue associated with the window that is to receive the pointer button information.

Default Processing

The frame control window procedure responds to this message by issuing the WinSetActiveWindow call and sets **result** to TRUE.

WM_BUTTON2DOWN

For the cause of this message, see "WM_BUTTON2DOWN" on page 12-9.

Parameters

For a description of the parameters, see "WM_BUTTON1DOWN" on page 12-7.

Remarks

This message is posted to the application queue associated with the window that is to receive the pointer button information.

Default Processing

The frame control window procedure processes this message identically to "WM_BUTTON1DOWN" on page 15-6.

WM_BUTTON1UP

For the cause of this message, see "WM_BUTTON1UP" on page 12-7.

Parameters

For a description of the parameters, see "WM_BUTTON1UP" on page 12-7.

Remarks

This message is posted to the application queue associated with the window that is to receive the pointer button information.

Default Processing

The frame control window procedure responds to this message by issuing the WinSetActiveWindow call and sets **result** to TRUE.

WM_BUTTON2UP

For the cause of this message, see "WM_BUTTON2UP" on page 12-9.

Parameters

For a description of the parameters, see "WM_BUTTON1UP" on page 12-7.

Remarks

This message is posted to the application queue associated with the window that is to receive the pointer button information.

Default Processing

The frame control window procedure processes this message identically to "WM_BUTTON1UP."

WM_CALCFRAMERECT

For the cause of this message, see "WM_CALCFRAMERECT" on page 12-12.

Parameters

For a description of the parameters, see "WM_CALCFRAMERECT" on page 12-12.

Remarks

Frame control calculates the appropriate rectangle, taking into account byte alignment, or nonbyte alignment if FCF_NOBYTEALIGN is specified.

Default Processing

The default window procedure takes no action on this message, other than to set **success** to FALSE.

WM_CHAR

This message is sent by controls to their owner window if they do not process the key stroke themselves. It is the most common means by which the input focus is switched around the various controls in a dialog box.

Parameters

For a description of the parameters, see "WM_CHAR" on page 12-14.

Default Processing

The frame control window procedure responds to this message as follows:

- If the message contains a valid VK_ value, that value is processed before any valid character in the message.
- If the character matches a mnemonic in the text of a button or static control child window, the focus is set to that window.
- If the character is tab or backtab, the focus is set to the next or previous tabstop window.
- If the character is up or left arrow, the focus is set to the previous item in the group.
- If the character is down or right arrow, the focus is set to the next item in the group.
- If the return key is pressed, a WM_COMMAND message is posted to itself, containing the identity of the button with the focus, or, if none, the identity of the default pushbutton.
- If the escape key is pressed, a WM_COMMAND message is posted to itself with the command value DID_CANCEL.

WM_CLOSE

For the cause of this message, see "WM_CLOSE" on page 12-16.

Parameters

For a description of the parameters, see "WM_CLOSE" on page 12-16.

Remarks

Frame control sends this message to the client window (FID_CLIENT) if it exists, otherwise it calls the WinDefWindowProc call.

Default Processing

The default window procedure posts a WM_QUIT message to the appropriate queue and sets **reply** to NULL.

WM_DRAWITEM

For the cause of this message, see "WM_DRAWITEM."

Parameters

For a description of the parameters, see "WM_DRAWITEM."

Remarks

The identity of the top-level action-bar menu that generated this message is found. If the identity is FID_MENU, the message is passed to the window with identity FID_CLIENT.

Default Processing

The default window procedure takes no action on this message, other than to set **reply** to NULL.

WM_ERASEBACKGROUND

This message causes a client window to be filled with the background, should this be appropriate.

Parameters

param1

hpsframe *(HPS)*
Presentation-space handle for the frame window.

param2

prcpaint *(PRECTL)*
Rectangle structure of rectangle to be painted.

This points to a *RECTL* structure.

Returns

reply

result *(BOOL)*
Processed indicator:

TRUE If a FID_CLIENT window exists, the area of the frame covered by the FID_CLIENT window is erased in the system-window background color.

If no FID_CLIENT window exists, the entire frame window is erased in the system-window background color.

FALSE The client window did process the message.

Remarks

The frame window procedure processes this message in the following manner:

1. The frame window sends this message to the client in response to the frame WM_PAINT message, with the presentation-space handle of the frame window (obtained from WinBeginPaint).
2. If the client window returns TRUE, the frame window procedure erases the rectangle of the frame window covered by the client window, by filling it with the system color SCLR_WINDOW.
3. If the client window returns FALSE, no action is taken. This is the default behavior, as WinDefWindowProc returns FALSE if passed this message.
4. Also, the client window can use the presentation-space handle passed in this message to selectively erase parts of the screen. If the client window processes the message in this way, FALSE should be returned to avoid the erasure being done automatically by the frame window procedure.

 It should be noted again that the presentation space is *not* a client window presentation space; it is a presentation space for the frame window returned by WinBeginPaint, that is, a cached presentation space in frame (not client) window coordinates, clipped to the area of the frame that needs to be updated (possibly including areas outside the client window).

Default Processing

The default window procedure takes no action on this message, other than to set **result** to FALSE.

WM_FLASHWINDOW

An application has issued a WinFlashWindow function.

Parameters

param1

flash *(BOOL)*
Flash indicator:

TRUE Start the window border flashing
FALSE Stop the window border flashing.

param2 *(BIT32)*
Reserved.

NULL Reserved value.

Returns

reply

result *(BOOL)*
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Default Processing

The frame control window procedure responds to this message from an application by starting or stopping the flashing of the window border, and by setting **result** as appropriate.

WM_FOCUSCHANGE

This message occurs when the window possessing the focus is changed.

Parameters

param1

focus *(HWND)*
Focus window handle.

param2

setfocus *(BOOL)*
Focus flag:

TRUE The window is receiving the focus and **Focus** identifies the window losing the focus
FALSE The window is losing the focus and **Focus** identifies the window receiving the focus.

focuschange *(BIT16)*
Focus changing indicators.

The indicators are passed from the WinFocusChange call with the exception of the FC_SETACTIVEFOCUS value, which is removed before this message is sent.

Returns

reply *(BIT32)*
Reserved.

NULL Reserved value.

Remarks

The frame control responds to this message by sending the other messages depending on the value of the **FocusChange** parameter. These messages, if sent, are sent in the following order:

1. WM_SETFOCUS to the window losing the focus.
2. WM_SETSELECTION to the windows losing their selection.
3. WM_ACTIVATE to the windows being deactivated.
4. WM_ACTIVATE to the windows being activated.
5. WM_SETSELECTION to the windows being selected.
6. WM_SETFOCUS to the window receiving the focus.

Default Processing

The default window procedure sends this message to either the owner, if one exists, or to the parent of the window, if it is not the desktop window, otherwise it sets **reply** to NULL.

WM_FORMATFRAME

This message is sent to a frame window to calculate the sizes and positions of all of the frame controls and the client window.

Parameters

param1

swp *(PSWP)*
Structure array.

This points to an array that is to hold the *SWP* structures.

param2

prectl *(PRECTL)*
Pointer to client window rectangle.

This is typically the window rectangle of **swp**, but where the window has a wide border, as specified by FCF_DLGBORDER for example, the rectangle is inset by the size of the border.

Returns

reply

count *(COUNT2)*
Count of the number of *SWP* arrays returned.

Remarks

Applications that subclass frame controls may find that the frame is already subclassed; the number of frame controls is variable.

The WM_FORMATFRAME and WM_QUERYFRAMECTLCOUNT messages must always be subclassed by calling the previous window procedure and modifying its result.

Default Processing

The *SWP* structure for the FID_CLIENT frame control, if present, is the last element of the **swp** parameter, unless additional frame controls are added by subclassing. The *SWP* structures for these follow that for FID_CLIENT, if present. The frame control window procedure first sends the message to the FID_CLIENT window. If FID_CLIENT returns **count** to indicate that the message has been processed, no additional processing is performed.

WM_INITMENU

For the cause of this message, see "WM_INITMENU" on page 17-4.

Parameters

For a description of the parameters, see "WM_INITMENU" on page 17-4.

Remarks

The identity of the top-level action-bar menu that generated this message is found. If the identity is FID_MENU, the message is passed to the window with identity FID_CLIENT. If the identity is FID_SYSMENU the system menu state is initialized according to the current state of the window.

Default Processing

The default window procedure takes no action on this message, other than to set **reply** to NULL.

WM_MEASUREITEM

For the cause of this message, see "WM_MEASUREITEM" on page 12-25.

Parameters

For a description of the parameters, see "WM_MEASUREITEM" on page 12-25.

Remarks

The identity of the top-level action bar menu that generated this message is found. If the identity is FID_MENU, the message is passed to the window with identity FID_CLIENT.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set **height** to the default value of NULL, which is equivalent to zero.

WM_MENUSELECT

For the cause of this message, see "WM_MENUSELECT" on page 17-6.

Parameters

For a description of the parameters, see "WM_MENUSELECT" on page 17-6.

Remarks

The identity of the top-level action-bar menu that generated this message is found. If the identity is FID_MENU, the message is passed to the window with identity FID_CLIENT.

Default Processing

The default window procedure takes no action on this message, other than to set **result** to TRUE.

WM_NEXTMENU

For the cause of this message, see "WM_NEXTMENU" on page 17-7.

Parameters

For a description of the parameters, see "WM_NEXTMENU" on page 17-7.

Remarks

The frame control window procedure processes the message by returning the handle of the system menu window if **menu** is the handle of the main action bar window, or by returning the handle of the main action bar window if **menu** is the handle of the system menu window.

Default Processing

The default window procedure takes no action on this message, other than to set **newmenu** to NULL.

WM_PAINT

For the cause of this message, see "WM_PAINT" on page 12-30.

Parameters

For a description of the parameters, see "WM_PAINT" on page 12-30.

Default Processing

The frame is redrawn as governed by the FCF_BORDER or FCF_DLGBORDER style. A WM_ERASEBACKGROUND message is sent to FID_CLIENT window, and if it returns FALSE, then the FID_CLIENT window is erased to the system-provided window background color and sets **reply** to NULL.

WM_QUERYBORDERSIZE

This message is sent to the frame window to determine the width and height of the window's border.

Parameters

param1

size *(PWPOINT)*
Width and height of size border control.

This points to a *WPOINT* structure, that is used to hold the width in the **x** parameter and the height in the **y** parameter.

param2 *(BIT32)*
Reserved.

NULL Reserved value.

Returns

reply

result *(BOOL)*
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Remarks

The frame window responds to this message by returning the width and height of its border in the **size** parameter, as follows:

SV_CX/CYSIZEBORDER If FCF_SIZEBORDER is specified
SV_CX/CYDLGFRAME If FCF_DLGBORDER is specified
SV_CX/CYBORDER If FS_BORDER is specified.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set **result** to the default value of NULL, which is equivalent to FALSE.

WM_QUERYCONVERTPOS

For the cause of this message, see "WM_QUERYCONVERTPOS" on page 12-33.

Parameters

For a description of the parameters, see "WM_QUERYCONVERTPOS" on page 12-33.

Remarks

The frame control window procedure returns QCP_NOCONVERT,

Default Processing

For the default window procedure processing of this message see "WM_QUERYCONVERTPOS" on page 12-33.

WM_QUERYFOCUSCHAIN

This message is used to request the handle of a window in the focus chain.

Parameters

param1

cmd *(BIT16)*
Command to be performed.

This field contains a flag to indicate what action is to be performed:

QFC_NEXTINCHAIN	Return the next window in the focus chain. The **parent** parameter is not used.
QFC_ACTIVE	Return the handle of the frame window that would be activated or deactivated, if this window gains or loses the focus. The window handle returned is a child of the window specified by the **parent** parameter.
QFC_FRAME	Return the handle of the first frame window associated with this window. The **parent** parameter is not used.
QFC_SELECTACTIVE	Return the handle of the window from the group of owned windows to which this window belongs which either currently has the focus or, if no window has the focus, previously had the focus. Return NULL, if no window in the owner group has had the focus. The **parent** parameter is not used.
QFC_PARTOFCHAIN	Return TRUE if the handle of the window receiving this message is **parent**, otherwise return false.

param2

parent *(HWND)*
Parent window.

Returns

reply

result *(HWND)*
Handle of window requested.

Remarks

The frame control responds to this message by returning:

For **cmd** set to QFC_NEXTINCHAIN:

- if a frame owner window exists, its focus save window (if it exists) or the window handle returned by the frame owner in response to this message.
- if a frame owner window does not exist, its parent.

For **cmd** set to QFC_FRAME:

- its own window handle.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set **result** to the default value of NULL.

WM_QUERYFRAMECTLCOUNT

This message is sent to the frame window in response to the receipt of a WM_SIZE or a WM_UPDATEFRAME message.

Parameters

param1 *(BIT32)*
Reserved.

NULL Reserved value.

param2 *(BIT32)*
Reserved.

NULL Reserved value.

Returns

reply

controlcount *(SHORT)*
Count of frame controls.

Remarks

By sending this message to itself, any procedures that subclass the frame window become aware that the number of frame controls is being calculated and include any special frame controls of the subclass in the count.

This count is used to allocate the appropriate number of *SWP* structures that are passed in the WM_FORMATFRAME message.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set **controlcount** to the default value of NULL which is equivalent to zero.

WM_QUERYFRAMEINFO

This message enables an application to query information about frame windows.

Parameters

param1 *(BIT32)*
Reserved.

NULL Reserved value.

param2 *(BIT32)*
Reserved.

NULL Reserved value.

Returns

reply

flags *(BIT32)*
Frame information flags.

One or more of the following are returned:

FI_FRAME	Identifies a frame window.
FI_OWNERHIDE	The frame window is hidden when its owner is hidden.
FI_NOMOVEWITHOWNER	The frame window does not move with its owner.
FI_ACTIVATEOK	The frame window may be activated. This means, for example, that the frame window is not disabled.

Remarks

This message can be used to query whether or not a particular window is a frame window.

Default Processing

The default window procedure takes no action on this message, other than to set **reply** to NULL.

WM_QUERYICON

This message is sent to a frame window to query its associated icon.

Parameters

param1 *(BIT32)*
Reserved.

NULL Reserved value.

param2 *(BIT32)*
Reserved.

NULL Reserved value.

Returns

reply

icon *(HPOINTER)*
Handle to the icon.

Default Processing

The icon for the frame is returned.

WM_QUERYWINDOWPARAMS

This message occurs when an application queries the frame control window parameters.

Parameters

For a description of the parameters, see "WM_QUERYWINDOWPARAMS" on page 12-35.

Default Processing

The frame control window procedure queries the appropriate window parameters in accordance with **wndparams** and sets **result** to TRUE if the operation is successful, otherwise to FALSE.

The window text of a frame control is obtained by sending this message to its FID_TITLEBAR.

WM_SETBORDERSIZE

This message is sent to the frame window to the change the width and height of the border.

Parameters

param1

cx *(USHORT)*
Width of border.

param2

cy *(USHORT)*
Height of border.

Returns

reply

result *(BOOL)*
Success indicator:

TRUE	Successful completion
FALSE	Error occurred.

Remarks

The frame control sets the width and height to **cx** and **cy** respectively.

Default Processing

The default window procedure takes no action on this message, other than to set **result** to FALSE.

WM_SETICON

This message is sent to a frame window to set its associated icon.

Parameters

param1

icon *(HPOINTER)*
New icon handle.

param2 *(BIT32)*
Reserved.

NULL	Reserved value.

Returns

reply

result *(BOOL)*
Success indicator:

TRUE	Successful completion
FALSE	Error occurred.

Default Processing

The icon for the frame is set.

WM_SETWINDOWPARAMS

This message occurs when an application sets or changes the frame control window parameters.

Parameters

For a description of the parameters, see "WM_SETWINDOWPARAMS" on page 12-40.

Default Processing

The frame control window procedure sets the appropriate window parameters in accordance with **wndparams** and sets **result** to TRUE if the operation is successful, otherwise to FALSE.

The window text of a frame control is set by sending this message to its FID_TITLEBAR.

WM_SIZE

For the cause of this message, see "WM_SIZE" on page 12-42.

Parameters

For a description of the parameters, see "WM_SIZE" on page 12-42.

Default Processing

The frame control window procedure responds to this message by sending a WM_FORMATFRAME message to itself and by setting **reply** to NULL.

WM_SYSCOMMAND

This message occurs when a control window has a significant event to notify to its owner, or when a key stroke has been translated by an accelerator table into a WM_SYSCOMMAND.

Parameters

param1

cmd *(USHORT)*
Command value.

The frame control takes the action described on these **cmd** values:

SC_SIZE	Sends a WM_TRACKFRAME to the frame window.
SC_MOVE	Sends a TBM_TRACKMOVE message to the control with the identifier FID_TITLEBAR.
SC_MINIMIZE	If a control with the identifier FID_MINMAX is present, minimizes the frame window, or restores it to a remembered size and position.
SC_MAXIMIZE	If a control with the identifier FID_MINMAX is present, maximizes the frame window, or restores it to a remembered size and position. When a window is moved or sized in the normal way at least one border should remain on the screen. When a window is maximized and the maximum size is as large as the screen, all borders should be positioned just outside the screen.
SC_RESTORE	If a control with the identifier FID_MINMAX is present, restores a maximized frame window to its previous size and position.
SC_NEXT	Cycles the active window status to the next main window.
SC_APPMENU	Sends a MM_STARTMENUMODE message to the control with the identifier FID_MENU.
SC_SYSMENU	Sends a MM_STARTMENUMODE message to the control with the identifier FID_SYSMENU.
SC_CLOSE	If Close is not enabled in the system menu, this message is ignored. Otherwise the frame posts a WM_CLOSE message to the client if it exists or to itself, if not.
SC_NEXTFRAME	The next frame window that is a child of the desktop window is activated.
SC_NEXTWINDOW	The next window with the same owner window is activated.
SC_TASKMANAGER	The Desktop Manager is activated.

SC_HELPEXTENDED The frame manager sends HM_EXT_HELP to the associated help manager Object Window. If there is no such associated window, the original message will be sent to client.

SC_HELPKEYS The frame manager sends HM_KEYS_HELP to the associated help manager Object Window. If there is no such associated window, the original message will be sent to the client.

SC_HELPINDEX The frame manager sends HM_HELP_INDEX to the associated help manager Object Window. If there is no such associated window, the original message will be sent to the client.

param2

source *(USHORT)*
Source type.

Identifies the type of control:

CMDSRC_PUSHBUTTON Posted by a pushbutton control. **cmd** is the window identifier of the pushbutton.

CMDSRC_MENU Posted by a menu control. **cmd** is the identifier of the menu item.

CMDSRC_ACCELERATOR Posted as the result of an accelerator. **cmd** is the accelerator command value.

CMDSRC_OTHER Other source. **cmd** gives further control-specific information defined for each control type.

pointer *(BOOL)*
Pointing-device indicator:

TRUE The message is posted as a result of a pointing-device operation

FALSE The message is posted as a result of a keyboard operation.

Returns

reply *(BIT32)*
Reserved.

NULL Reserved value.

Remarks

This message is posted to the window procedure of the owner of the frame control. **reply** is set to NULL.

Default Processing

The default window procedure takes no action on this message, other than to set **reply** to NULL.

WM_TRACKFRAME

This message is sent to a frame window whenever it is to be moved or sized.

Parameters

param1

trackflags *(BIT16)*
Tracking flags.

Contains a combination of one or more TF_* flags; for details, see the *TRACKINFO* data structure.

param2 *(BIT32)*
Reserved.

NULL Reserved value.

Returns

reply

result *(BOOL)*
Success indicator:

TRUE	Successful completion
FALSE	Error occurred, or the operation is terminated.

Remarks

The frame control window procedure responds to this message by causing a tracking rectangle to be drawn to move or size the window. For information, see the WinTrackRect call.

Default Processing

The default window procedure takes no action on this message, other than to set **result** to TRUE.

WM_TRANSLATEACCEL

For the cause of this message, see "WM_TRANSLATEACCEL" on page 12-47.

Parameters

For a description of the parameters, see "WM_TRANSLATEACCEL" on page 12-47.

Remarks

The frame control window procedure processes the message by checking whether the character is in the accelerator table, by using the WinTranslateAccel call.

Default Processing

The default window procedure takes no action on this message, other than to set **translated** to FALSE.

WM_TRANSLATEMNEMONIC

For the cause of this message, see "WM_TRANSLATEMNEMONIC" on page 12-47.

Parameters

For a description of the parameters, see "WM_TRANSLATEMNEMONIC" on page 12-47.

Remarks

The frame control window procedure processes the message by sending it to the application menu window, that is, the window with the identity FID_MENU.

Default Processing

For the default window procedure processing of this message, see "WM_TRANSLATEMNEMONIC" on page 12-47.

WM_UPDATEFRAME

This message is sent by an application after frame controls have been added or removed from the window frame.

Parameters

param1

createflags *(BIT32)*
Frame creation flags.

Contains the FCF_* flags that indicate which frame controls have been added or removed.

param2 *(BIT32)*
Reserved.

NULL	Reserved value.

Returns

reply

result *(BOOL)*
Processed indicator:

TRUE	Message processed
FALSE	Message ignored.

Remarks

This message must be sent to the frame window whenever an application adds or removes one of the frame controls identified by the FCF_* flags. It must also be sent if the application adds or removes a submenu of the frame window's menu bar.

The frame control window procedure first sends the message on to the FID_CLIENT window. If the FID_CLIENT window may either reformat the frame window and set **result** to TRUE, in which case the frame control window procedure takes no further action, or it may set **result** to FALSE, in which case the frame control window procedure performs the reformatting.

If **createflags** contains FCF_SIZEBORDER, reformatting the frame window includes invalidating the area occupied by the size border.

The frame control window procedure sets **result** to TRUE.

Programming Note: As this message causes any redrawing that is necessary, you must ensure that no drawing takes place when you actually add or remove a frame control, to prevent unnecessary redrawing. If you use WinSetParent, this is done by setting the **Redraw** to FALSE.

Default Processing

The default window procedure takes no action on this message, other than to set **reply** to NULL.

Chapter 16. List Box Control Window Processing

This system-provided window procedure processes the actions on a list box control (WC_LISTBOX).

List Box Control Styles

A list box can have these styles:

LS_HORZSCROLL The list box control enables the operator to scroll the list box horizontally.

LS_MULTIPLESEL The list box control enables the operator to select more than one item at any one time. Lists that do not have this style allow only a single selection at any one time.

LS_OWNERDRAW The list box control has one or more items that can be drawn by the owner. Typically, these items are represented by bit maps rather than by text strings.

LS_NOADJUSTPOS If this style is included, the list box control is drawn at the size specified. This can cause parts of an item to be shown.

LS_NOVERTSCROLL No vertical scroll bar is to be displayed.

List Box Control Data

None.

List Box Control Notification Messages

These messages are initiated by the list box control window to notify its owner of significant events.

WM_CONTROL

For the cause of this message, see "WM_CONTROL" on page 12-17.

Parameters

For a description of the parameters, see "WM_CONTROL" on page 12-17.

param1

id *(IDENTITY)*
Control-window identity.

notifycode *(USHORT)*
Notify code.

The list box control window procedure uses these notification codes:

LN_ENTER	Either the ENTER or RETURN key has been pressed while the list box control has the focus, or the list box control has been double-clicked.
LN_KILLFOCUS	The list box control loses the focus.
LN_SCROLL	The list box control is about to scroll horizontally. This can happen when the application has issued a WinScrollWindow call.
LN_SETFOCUS	The list box control receives the focus.
LN_SELECT	An item is being selected (or deselected).

Programming Note: To discover the index of the selected item, the application must use the LM_QUERYSELECTION message.

param2

controlspec *(HWND)*
List box control window handle.

Returns

reply *(BIT32)*
Reserved.

NULL Reserved value.

Remarks

The list box control window procedure generates this message and sends it to its owner, informing the owner of this event.

Default Processing

The default window procedure takes no action on this message, other than to set **reply** to NULL.

WM_DRAWITEM

This notification is sent to the owner of a list box control each time an item is to be drawn.

Parameters

param1

listbox *(IDENTITY)*
Window identifier.

The window identity of the list box control sending this notification message.

param2

owneritem *(POWNERITEM)*
Owner-item structure.

This points to an owner-item structure; see OWNERITEM on page 2-25.

Returns

reply

drawn *(BOOL)*
Item-drawn indicator:

TRUE	The owner draws the item, so the list box control does not draw it
FALSE	If the item contains text and the owner does not draw the item, the owner returns this value, and the list box control draws the item.

Remarks

The list box control window procedure only draws items that are represented by text strings and emphasizes selected items by inverting them.

If an application uses list box controls containing items that are not represented by text strings, or requires that the emphasized state of an item is to be drawn in a special manner, the list box control must specify the style LS_OWNERDRAW and those items must be drawn by the owner.

The list box control window procedure generates this message and sends it to the owner of the list box control, informing the owner that an item is to be drawn, offering the owner the opportunity to draw that item, and indicating that either the item has been drawn, or that the list box control is to draw it.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set **drawn** to the default value of NULL, which is equivalent to FALSE.

WM_MEASUREITEM

This notification is sent to the owner of a list box control to establish the height and width for an item in that control.

Parameters

param1

listbox *(SHORT)*
List-box identifier.

param2

itemIndex *(SHORT)*
Item index.

The zero-based index of the item which has changed.

Returns

reply

height *(SHORT)*
Height of item.

width *(SHORT)*
Width of item.

This value is required only if the list box control is scrollable horizontally, that is, it has a style of LS_HORZSCROLL.

Remarks

This message is sent every time an item changes in the list box. The list box control maintains the maximum height and width of the items and sets the scroll bar ranges.

All items in a list box must have the same height, which must be greater than or equal to the height of the current font.

In particular, this notification is sent to the owner of a list box that has a style of LS_OWNERDRAW, to offer the owner an opportunity to establish the height and width (for a horizontally scrollable list box control) of an item that accommodates any special requirements for the drawing of items in that list box.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set **height** to the default value of NULL, which is equivalent to zero.

List Box Control Window Messages

This section describes the list box control window procedure actions on receiving the following messages.

LM_DELETEALL

This message is sent to a list box control to delete all the items in the list box.

Parameters

param1 *(BIT32)*
Reserved.

NULL Reserved value.

param2 *(BIT32)*
Reserved.

NULL Reserved value.

Returns

reply

success *(BOOL)*
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Remarks

The list box control window procedure responds to this message by deleting all the items in the list box and by setting **success** to TRUE.

Default Processing

The default window procedure does not expect to receive this message and, therefore, takes no action on it, other than to set **success** to the default value of NULL, which is equivalent to FALSE.

LM_DELETEITEM

This message deletes an item from the list box control.

Parameters

param1

itemIndex *(SHORT)*
Item index.

The zero-based index of the item to be deleted.

param2 *(BIT32)*
Reserved.

NULL Reserved value.

Returns

reply

itemsleft *(SHORT)*
Number remaining.

The number of items in the list after the item is deleted.

Remarks

The list box control window procedure responds to this message by deleting the indexed item of the list box and by setting **itemsleft** to the count of the items in the list after the item is deleted.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set **itemsleft** to the default value of NULL, which is equivalent to zero.

LM_INSERTITEM

This message inserts an item into a list box control.

Parameters

param1

itemindex *(SHORT)*
Item index:

LIT_END	Add the item to the end of the list.
LIT_SORTASCENDING	Insert the item into the list sorted in ascending order.
LIT_SORTDESCENDING	Insert the item into the list sorted in descending order.
Other	Insert the item into the list at the offset specified by this zero-based index.

param2

itemtext *(PSTRL)*
Item text.

This points to the item text.

Returns

reply

indexinserted *(SHORT)*
Index of inserted item:

LIT_MEMERROR	The list box control cannot allocate space to insert the list item in the list.
LIT_ERROR	An error, other than LIT_MEMERROR, occurred.
Other	The zero-based index of the offset of the item within the list.

Remarks

The list box control window procedure responds to this message by inserting the item text identified by the **itemtext** parameter into the position in the list specified by the **itemindex** parameter.

The sorting sequence used is that defined by the WinCompareStrings call.

The list box control sets **indexinserted** to the zero-based index of the offset of the item within the list.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set **indexinserted** to the default value of NULL, which is equivalent to zero.

LM_QUERYCURSOR

This message determines the index of the item associated with the cursor of the list box control.

Parameters

param1 *(BIT32)*
Reserved

NULL Reserved value.

param2 *(BIT32)*
Reserved.

NULL Reserved value.

Returns

reply

itemcursor *(USHORT)*
Index of the item currently associated with the cursor.

LIT_NONE No item is associated with the cursor.
Other Index of the item currently associated with the cursor.

Remarks

The list box control window procedure responds to this message by returning the index of the item associated with the cursor.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set **itemcursor** to the default value of NULL, which is equivalent to zero.

LM_QUERYITEMCOUNT

This message returns a count of the number of items in the list box control.

Parameters

param1 *(BIT32)*
Reserved.

NULL Reserved value.

param2 *(BIT32)*
Reserved.

NULL Reserved value.

Returns

reply

itemcount *(SHORT)*
Item count.

Remarks

The list box control window procedure responds to this message by setting **itemcount** to the number of items in the list.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set **itemcount** to the default value of NULL, which is equivalent to zero.

LM_QUERYITEMHANDLE

This message returns the handle of the indexed item of the list box control.

Parameters

param1

itemIndex *(USHORT)*
Item index.

param2 *(BIT32)*
Reserved.

NULL Reserved value.

Returns

reply

result *(ULONG)*
Item handle.

Remarks

The meaning of the item handle is defined by the application. For example, it may be a pointer to an application defined data structure. Item handles are initialized to NULL when an item is created.

The list box control window procedure responds to this message by setting **result** to the handle of the item whose index is specified by **itemIndex**.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set **result** to the default value of NULL.

The item handle is initialized to zero.

LM_QUERYITEMTEXT

This message returns the text of the specified list box item.

Parameters

param1

itemIndex *(USHORT)*
Item index.

maxcount *(SHORT)*
Maximum count:

0 No text is copied.
Other Copy the item text as a null-terminated string, but limit the number of characters copied, including the null termination character, to this value.

param2

itemtext *(PSTRL)*
Buffer into which the item text is to be copied.

This points to a *STRL*.

Returns

reply

textlength *(SHORT)*
Length of item text.

The length of the text string, excluding the null termination character.

Remarks

The list box control window procedure responds to this message by copying up to **maxcount** characters, as a null-terminated string, from the text of the item specified by **itemindex** into the buffer identified by **itemtext**.

The length of the item text can be determined by using the LM_QUERYITEMTEXTLENGTH message.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set **reply** to the default value of NULL, which is equivalent to zero.

LM_QUERYITEMTEXTLENGTH

This message returns the length of the text of the specified list box item.

Parameters

param1

itemindex *(USHORT)*
Item index.

param2 *(BIT32)*
Reserved.

NULL Reserved value.

Returns

reply

textlength *(SHORT)*
Length of item text.

The length of the text string, excluding the null termination character.

LIT_ERROR Error occurred. For example, the item specified by its index does not exist.
Other Length of item text.

Remarks

The list box control window procedure responds to this message by setting **textlength** to the length in characters of the text of the item specified by **itemindex**.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than set **textlength** to the default value of NULL, which is equivalent to zero.

LM_QUERYSELECTION

This message is used to enumerate the selected item, or items, in a list box.

Parameters

param1

itemstart *(SHORT)*
Index of the start item.

If the list box allows multiple selected items, that is, it has a style of LS_MULTIPLESEL, then this parameter indicates the index of the item from which the search for the next selected item is to begin. Therefore, to get all the selected items of the list, this message is sent repeatedly, each time setting this parameter to the index of the item returned by the previous usage of this message.

If the list box only allows a single selection, this parameter is ignored.

LIT_FIRST Start the search at the first item.
Other Start the search after the item specified by this index.

param2 *(BIT32)*
Reserved.

NULL Reserved value.

Returns

reply

itemselected *(USHORT)*
Index of the selected item:

LIT_NONE No selected item.

For a single selection list box, this implies that there is no selected item in the list box. For a multiple selection list box, this implies that there is no selected item in the list box whose index is higher than the index specified by the **itemstart** parameter.

Other Index of selected item. For a single selection list box, this is the index of the only selected item in the list box. For a multiple selection list box, this is the index of the next selected item in the list box whose index is higher than the index specified by the **itemstart** parameter.

Remarks

The list box control window procedure responds to this message by returning in **itemselected** the zero-based index of the selected item or next selected item after **itemstart**, if any.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than set **itemselected** to the default value of NULL, which is equivalent to zero.

LM_QUERYTOPINDEX

This message obtains the index of the item currently at the top of the list box.

Parameters

param1 *(BIT32)*
Reserved.

NULL Reserved value.

param2 *(BIT32)*
Reserved.

NULL Reserved value.

Returns

reply

itemtop *(SHORT)*
Index of the item currently at the top of the list box:

LIT_NONE No items in the list box
Other Index of the item currently at the top of the list box.

Remarks

The list box control window procedure responds to this message by returning in **itemtop** the zero-based index of the item currently at the top of the list box.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set **itemtop** to the default value of NULL, which is equivalent to zero.

LM_SEARCHSTRING

This message returns the index of the list box item whose text matches the string.

Parameters

param1

cmd *(USHORT)*
Command.

Defines the criteria by which the string specified by the **searchstring** parameter is to be compared with the text of the items, to determine the index of the first matching item.

These values can be combined using the logical-OR operator:

LSS_CASESENSITIVE	Matching occurs if the item contains the characters specified by the **searchstring** parameter exactly. **This value is mandatory.**
LSS_PREFIX	Matching occurs if the leading characters of the item contain the characters specified by the **searchstring** parameter. If this value is specified, LSS_SUBSTRING must not be specified.
LSS_SUBSTRING	Matching occurs if the item contains a substring of the characters specified by the **searchstring** parameter. If this value is specified, LSS_PREFIX must not be specified.

itemstart *(USHORT)*
Index of the start item:

LIT_FIRST	Start the search at the first item.
Other	Start the search after the item specified by this index.

Therefore, to get all the items in the list whose text matches the string, this message is sent repeatedly, each time setting this parameter to the index of the item returned by the previous usage of this message.

param2

searchstring *(PSTRL)*
Search string.

This points to a *STRL*.

Returns

reply

itemmatched *(USHORT)*
Index item whose text matches the string:

LIT_ERROR	Error occurred
LIT_NONE	No item found
Other	Index item whose text matches the string.

Remarks

The list box control window procedure responds to this message by setting **itemmatched** to the index of the next item whose text matches the string specified by **searchstring**.

All the items of the list are searched until a match is found, that is, the search wraps from the end to the start of the list.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set **itemmatched** to the default value of NULL, which is equivalent to zero.

LM_SELECTITEM

This message is used to set the selection state of an item in a list box.

Parameters

param1

itemIndex *(SHORT)*
Index of the item to be selected or deselected:

LIT_NONE	All items are to be deselected
Other	Index of the item to be selected or deselected.

param2

select *(BOOL)*
Select flag:

Ignored if **itemIndex** is set to LIT_NONE.

TRUE	The item is selected. If the control is a single selection list box (that is, it does not have the style of LS_MULTIPLESEL), any previously selected item is deselected.
FALSE	The item is deselected.

Returns

reply

success *(BOOL)*
Success indicator:

TRUE	Successful completion
FALSE	Error occurred. For example, when the item does not exist in the list box, or when an item that is not selected is deselected.

Remarks

The list box control window procedure responds to this message by setting the selection state, as indicated by **select**, of the item whose index is specified in **itemIndex**.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set **success** to the default value of NULL, which is equivalent to FALSE.

LM_SETCURSOR

This message associates the cursor with a specified item of the list box control.

Parameters

param1

itemIndex *(SHORT)*
Item index:

LIT_NONE	Disassociate the cursor from the list box control.
Other	Associate the cursor with the item in the list box control at the offset specified by this zero-based index.

param2 *(BIT32)*
Reserved.

NULL	Reserved value.

Returns

reply

changed *(BOOL)*
Changed indicator:

TRUE	The index of the item in the list box control associated with the cursor has been changed
FALSE	The index of the item in the list box control associated with the cursor has not been changed.

Remarks

The list box control window procedure responds to this message by associating the cursor with the item of the list box whose index is specified, replacing any previous item which was associated with the cursor.

This message does not affect the selection in the list box control.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set **changed** to the default value of NULL, which is equivalent to FALSE.

LM_SETITEMHANDLE

This message sets the handle of the specified list box item.

Parameters

param1

itemindex *(USHORT)*
Item index.

param2

itemhandle *(ULONG)*
Item handle.

Returns

reply

success *(BOOL)*
Success indicator:

TRUE	Successful completion
FALSE	Error occurred.

Remarks

The list box control window procedure responds to this message by setting the handle of the item whose index is specified by **itemindex** to the value specified by **itemhandle**.

Default Processing

The meaning of the item handle is defined by the application. It may, for example, be a pointer to an application defined data structure.

Item handles are initialized to NULL when an item is created.

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set **success** to the default value of NULL, which is equivalent to FALSE.

LM_SETITEMHEIGHT

This message sets the height of the items in a list box.

Parameters

param1

newheight *(BIT32)*
Height of items in list box.

param2 *(BIT32)*
Reserved.

NULL Reserved value.

Returns

reply

success *(BOOL)*
Success indicator:

TRUE Successful operation
FALSE Error occurred.

Remarks

The list box control window procedure responds to this message by setting the height of the items in a list box to that specified by **newheight**.

This message does **not** send a WM_MEASUREITEM message.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set **success** to the default value of NULL, which is equivalent to FALSE.

LM_SETITEMTEXT

This message sets the text into the specified list box item.

Parameters

param1

itemIndex *(SHORT)*
Item index.

param2

itemtext *(PSTRL)*
Item text.

This points to a *STRL*.

Returns

reply

success *(BOOL)*
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Remarks

The list box control window procedure responds to this message by copying the text identified by the **itemtext** parameter into the item in the list specified by the **itemIndex** parameter.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set **success** to the default value of NULL, which is equivalent to FALSE.

LM_SETSELECTION

This message sets the selection state of the specified item of the list box control.

Parameters

param1

itemIndex *(USHORT)*
Item index.

param2

selected *(BOOL)*
Selection indicator:

TRUE	Select the item
FALSE	Do not select the item.

Returns

reply

changed *(BOOL)*
Changed indicator:

TRUE	The selection state of the item has been changed
FALSE	The selection state of the item has not been changed.

Remarks

The list box control window procedure responds to this message by setting the selection state of the specified item.

This message does not affect any item associated with the cursor.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set **changed** to the default value of NULL, which is equivalent to FALSE.

LM_SETTOPINDEX

This message is used to scroll a particular item to the top of the list box.

Parameters

param1

itemIndex *(SHORT)*
Index of the item to be made top.

param2 *(BIT32)*
Reserved.

NULL	Reserved value.

Returns

reply

success *(BOOL)*
Success indicator:

TRUE	Successful completion
FALSE	Error occurred.

Remarks

The list box control window procedure responds to this message by scrolling the item whose index is identified by **itemindex** to the top of the list box.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set **success** to the default value of NULL, which is equivalent to FALSE.

WM_CHAR

For the cause of this message, see "WM_CHAR" on page 12-14.

Parameters

For a description of the parameters, see "WM_CHAR" on page 12-14.

Remarks

The list box control window procedure responds to this message by sending it to its owner if it has not processed the key stroke. This is the most common means by which the input focus is switched around the various controls in a dialog box.

The key strokes processed by a list box control are:

down arrow Moves the selection down one item, scrolling the list box by one item, if necessary, to make the next item visible. When the selection reaches the bottom, the down arrow has no effect.

up arrow Moves the selection up one item, scrolling the list box by one item, if necessary, to make the previous item visible. When the selection reaches the top, the up arrow has no effect.

page down Moves the selection down "one page" scrolling the list box by the number of items visible in the list box.

For example, if the list box displays seven items and the item number 1 is selected and positioned at the top of the list box, pressing the page down key causes item number 8 to be selected and displayed at the top of the list box. Pressing page down when the last item is selected has no effect.

page up Moves the selection up "one page" scrolling the list box by the number of items visible in the list box.

For example, if the list box displays seven items and the item number 8 is selected and positioned at the top of the list box, pressing the page up key causes item number 1 to be selected and displayed at the top of the list box. Pressing the page up key when the first item is selected has no effect.

Default Processing

The default window procedure takes no action on this message, other than to set **reply** to NULL.

WM_QUERYCONVERTPOS

For the cause of this message, see "WM_QUERYCONVERTPOS" on page 12-33.

Parameters

For a description of the parameters, see "WM_QUERYCONVERTPOS" on page 12-33.

Remarks

The list box control window procedure returns QCP_NOCONVERT,

Default Processing

For the default window procedure processing of this message see "WM_QUERYCONVERTPOS" on page 12-33.

WM_QUERYWINDOWPARAMS

Occurs when an application queries the list box control window parameters.

Parameters

For a description of the parameters, see "WM_QUERYWINDOWPARAMS" on page 12-35.

Remarks

The list box control window procedure responds to this message by passing it to the default window procedure.

Default Processing

The default window procedure sets the **length**, **presparamslength**, and **ctldatalength** parameters of the *WNDPARAMS* data structure, identified by **wndparams**, to zero and sets **result** to FALSE.

WM_SETWINDOWPARAMS

This message occurs when an application sets or changes the list box control window parameters.

Parameters

For a description of the parameters, see "WM_SETWINDOWPARAMS" on page 12-40.

Remarks

The list box control window procedure responds to this message by passing it to the default window procedure.

Default Processing

The default window procedure takes no action on this message, other than to set **result** to FALSE.

Chapter 17. Menu Control Window Processing

This system-provided window procedure processes the actions on a menu control (WC_MENU).

Menu Control Styles

Menu Control Style	
MS_ACTIONBAR	The items in the list are displayed side-by-side. This style is used to implement a top level menu. Menus that do not have this style are displayed in one or more columns and are submenus associated with an action bar. All menu controls have styles CS_SYNCPAINT and CS_PARENTCLIP.
MS_TITLEBUTTON	Used to identify menus that can be used as buttons in the title bar. Can only be used with MS_ACTIONBAR. This style causes the menu to be drawn using the Common User Access (CUA) colors specified for the title bar rather than the action bar.
MS_VERTICALFLIP	Normally, pull-down menus (the default, without the MS_VERTICALFLIP style) are displayed below their associated action bar item. If there is not room on the screen to display the entire pull-down in this manner, and if there is room to display the pull-down above the action bar, it is displayed above the action bar. Pull-down menus with the MS_VERTICALFLIP style are flipped vertically. That is, they are displayed above the menu if possible, otherwise below it. The vertical flip style must be set explicitly by the application when the window is minimized, and must be reset when it is restored. If an application action bar contains this style, the style is applied to all pull-down menus belonging to the action bar (the style does not directly affect the display of the action bar). This provides a convenient means for the application to flip the appearance of all pull-down menus.

Menu Item Styles

Menu Item Style	
MIS_SUBMENU	The item is a submenu. When the user selects this type of item, a submenu is displayed from which the user must make further selection. Items that are not submenu items are command items.
MIS_SEPARATOR	The display object is a horizontal dividing line. This type of item can only be used in pull-down menus. This type of item cannot be enabled, checked, disabled, highlighted, or selected by the user. The functional object is NULL when this style is specified.
MIS_BITMAP	The display object is a bit map.
MIS_TEXT	The display object is a text string.
MIS_BUTTONSEPARATOR	The item is a menu button. Any menu can have zero, one, or two items of this type. These are the last items in a menu and are automatically displayed after a separator bar. The user cannot move the cursor to these items, but can select them with the pointing device or with the appropriate key.
MIS_BREAK	The item is the last one in a row or column. The next item is displayed at the beginning of a new row or column.
MIS_BREAKSEPARATOR	The same as MIS_BREAK, except that it draws a separator between rows or columns of a pull-down menu. This style can only be used within a submenu.
MIS_SYSCOMMAND	If this item is selected, the menu notifies the owner by posting a WM_SYSCOMMAND message rather than a WM_COMMAND message.
MIS_OWNERDRAW	Items with this style are drawn by the owner. WM_DRAWITEM and WM_MEASUREITEM notification messages are sent to the owner to draw the item or determine its size.
MIS_HELP	If the item is selected, the menu notifies the owner by posting a WM_HELP message rather than a WM_COMMAND message.
MIS_STATIC	This type of item exists for information purposes only. It cannot be selected with the pointing device or keyboard.

Menu Item Attributes

Menu items have the following attributes.

Applications can get and set the state of these attributes by sending MM_QUERYITEMATTR and MM_SETITEMATTR messages.

MIA_HILITED	The state of this attribute is TRUE, if and only if, the item is selected.
MIA_CHECKED	If this attribute is TRUE a checkmark appears next to the item.
MIA_DISABLED	This attribute is TRUE if the item is disabled and cannot be selected. The item is drawn in a disabled state.
MIA_FRAMED	If this attribute is TRUE a frame is drawn around the item.
MIA_NODISMISS	If this item is selected, the pull-down menu containing this item should not be hidden before notifying the application window of the selection.

Menu Control Notification Messages

These messages are initiated by the menu control window procedure to notify its owner of significant events.

WM_COMMAND

For the cause of this message, see "WM_COMMAND" on page 12-16.

Parameters

For a description of the parameters, see "WM_COMMAND" on page 12-16.

The menu control window procedure sets **cmd** to the menu-item identity.

Remarks

The menu control window procedure generates this message and posts it to the queue of its owner, when an item is selected that does not have the style of MIS_SYSCOMMAND or MIS_HELP, but only if the WM_MENUSELECT message returns a **result** of TRUE.

Default Processing

The default window procedure takes no action on this message, other than to set **reply** to NULL.

WM_DRAWITEM

This notification is sent to the owner of a menu control each time an item is to be drawn.

Parameters

param1

menu *(IDENTITY)*
Window identifier.

The window identity of the menu control sending this notification message.

param2

ownerItem *(POWNERITEM)*
Owner-item structure.

This points to an owner-item structure; see OWNERITEM on page 2-25.

Returns

reply

drawn *(BOOL)*
Item-drawn indicator:

TRUE	The owner draws the item, and so the menu control does not draw it
FALSE	If the item contains text and the owner does not draw the item, the owner returns this value and the menu control draws the item.

Remarks

The menu control window procedure only draws items that are represented by text strings and emphasizes selected items by inverting them.

If an application uses menu controls containing items that are not represented by text strings, or requires that the emphasized state of an item is to be drawn in a special manner, then the menu control must specify the style MIS_OWNERDRAW and those items must be drawn by the owner.

The menu control window procedure generates this message and sends it to its owner, informing the owner that an item is to be drawn, offering the owner the opportunity to draw that item, and to indicate that either the item has been drawn, or that the menu control is to draw it.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set **drawn** to the default value of NULL which is equivalent to FALSE.

WM_HELP

For the cause of this message, see "WM_HELP" on page 12-22.

Parameters

For a description of the parameters, see "WM_HELP" on page 12-22.

The menu control window procedure sets **cmd** to the menu-item identity.

Remarks

This message is identical to a WM_COMMAND message, but implies that the application should respond to this message by displaying help information.

The menu control window procedure generates this message and posts it to the queue of its owner when an item is selected that has the style of MIS_HELP, but only if WM_MENUSELECT returns a **result** of TRUE.

Default Processing

The default window procedure sends this message to the parent window, if it exists and is not the desktop. Otherwise, it sets **reply** to NULL.

WM_INITMENU

This message occurs when a menu control is about to become active.

Parameters

param1

menuId *(SHORT)*
Menu-control identifier.

param2

hwnd *(HWND)*
Menu-window handle.

Returns

reply *(BIT32)*
Reserved.

NULL Reserved value.

Remarks

It offers the owner the opportunity to perform some initialization on the menu items before they are presented.

The menu control window procedure generates this message and sends it to its owner, informing the owner of the event.

Default Processing

The default window procedure takes no action on this message, other than to set **reply** to NULL.

WM_MEASUREITEM

This notification is sent to the owner of a menu control to establish the height for an item in that control.

Parameters

param1

menu *(SHORT)*
Menu identifier.

param2

owneritem *(POWNERITEM)*
Owner-item structure.

This points to an *OWNERITEM* structure.

Returns

reply

height *(SHORT)*
Height of item.

Remarks

This message is only sent at the time the menu control is created. When the owner receives this message, it must calculate and return the height of an item to the control.

All items in a menu must have the same height, and that must be greater than or equal to the height of the current font.

In particular, this notification is sent to the owner of a menu that has a style of MIS_OWNERDRAW, to offer the owner an opportunity to establish the height of an item that accommodates any special requirements for the drawing of items in that menu.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set **height** to the default value of NULL, which is equivalent to zero.

WM_MENUEND

This message occurs when a menu control is about to terminate.

Parameters

param1

menuId *(USHORT)*
Menu-control identifier.

param2

hwnd *(HWND)*
Menu control window handle.

Returns

reply *(BIT32)*
Reserved.

NULL Reserved value.

Remarks

The menu control window procedure generates this message and sends it to its owner, informing the owner of this event.

Default Processing

The default window procedure takes no action on this message, other than to set **reply** to NULL.

WM_MENUSELECT

This message occurs when a menu item has been selected.

Parameters

param1

item *(USHORT)*
Identifier of selected item.

postcommand *(BOOL)*
Post-command flag:

TRUE Indicates that either a WM_COMMAND, WM_SYSCOMMAND, or WM_HELP message is being posted by the menu control on return from the owner, subject to **result**.

FALSE Indicates that no message is being posted by the menu control on return from the owner, subject to **result**.

param2

hwnd *(HWND)*
Menu control window handle.

Returns

reply

result *(BOOL)*
Post indicator:

TRUE Indicates that either a WM_COMMAND, WM_SYSCOMMAND, or WM_HELP message is to be posted by the menu control window procedure. The menu is dismissed if the selected item does not have a style of MIA_NODISMISS.

FALSE Indicates that no message is to be posted by the menu control window procedure and that the menu is not dismissed.

Remarks

The menu control window procedure generates this message and sends it to its owner, informing the owner of this event.

When the message is returned from its owner, menu control acts on **result** as appropriate.

It must not be posted to the menu control.

Default Processing

The default window procedure takes no action on this message, other than to set **result** to TRUE.

WM_NEXTMENU

This message occurs when either the beginning or the end of the menu is reached by using the cursor control keys.

Parameters

param1

menu *(HWND)*
Menu control window handle.

param2

prev *(BOOL)*
Previous menu indicator:

TRUE	Beginning of the menu has been reached
FALSE	End of the menu has been reached.

Returns

reply

newmenu *(HWND)*
New menu window handle:

NULL	No new menu
Other	New menu window handle.

Remarks

The menu control generates this message and sends it to its owner, informing the owner of this event.

Default Processing

The default window procedure takes no action on this message, other than to set **newmenu** to NULL.

WM_SYSCOMMAND

For the cause of this message, see "WM_SYSCOMMAND" on page 12-43.

Parameters

For a description of the parameters, see "WM_SYSCOMMAND" on page 12-43.

The menu control window procedure sets **cmd** to the menu-item identity.

Remarks

The menu control window procedure generates this message and posts it to the queue of its owner, when an item is selected that has the style of MIS_SYSCOMMAND, but only if the WM_MENUSELECT message returns a **result** of TRUE.

Default Processing

The default window procedure takes no action on this message, other than to set **reply** to NULL.

Menu Control Window Messages

This section describes the menu control window procedure actions on receiving the following messages.

MM_DELETEITEM

This message deletes a menu item.

Parameters

param1

item *(USHORT)*
Item identifier.

includesubmenus *(BOOL)*
Include submenus indicator:

TRUE If the menu does not have an item with the specified identifier, search the submenus and subdialogs of the menu for an item with the specified identifier and delete it.

FALSE If the menu does not have an item with the specified identifier, do not search the submenus and subdialogs of the menu for an item with the specified identifier.

param2 *(BIT32)*
Reserved.

NULL Reserved value.

Returns

reply

itemsleft *(SHORT)*
Number remaining.

The number of items in the menu after the item is deleted.

Remarks

The menu control window procedure responds to this message by deleting the identified item from the menu or its submenus.

Programming Note: It must be sent, not posted, to the menu control.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set **itemsleft** to the default value of NULL, which is equivalent to zero.

MM_DISMISSMENU

This message is used to dismiss a menu, that is, to make it invisible.

Parameters

param1 *(BIT32)*
Reserved.

NULL Reserved value.

param2 *(BIT32)*
Reserved.

NULL Reserved value.

Returns

reply

success *(BOOL)*
Success indicator:

TRUE	Successful completion
FALSE	Error occurred.

Remarks

The menu control window procedure responds to this message by dismissing the menu.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set **success** to the default value of NULL, which is equivalent to FALSE.

MM_ENDMENUMODE

This message is sent to a menu control to terminate menu selection.

Parameters

param1

dismiss *(BOOL)*
Dismiss menu indicator:

TRUE	Dismiss the submenu or subdialog window
FALSE	Do not dismiss the submenu or subdialog window.

param2 *(BIT32)*
Reserved.

NULL	Reserved value.

Returns

reply *(BIT32)*
Reserved.

NULL	Reserved value.

Remarks

The menu control window procedure responds to this message by terminating menu selection.

Programming Note: It must be sent, not posted, to the menu control.

Default Processing

The default window procedure does not expect to receive this message and, therefore, takes no action on it, other than to set **reply** to the default value of NULL which is equivalent to NULL.

MM_INSERTITEM

This message inserts a menu item into a menu.

Parameters

param1

menuitem *(PMENUITEM)*
Menu-item data structure.

This points to a *MENUITEM* structure.

param2

itemtext *(PSTRL)*
Item text.

Returns

reply

indexInserted *(SHORT)*
Index of inserted item:

MIT_MEMERROR	The menu control cannot allocate space to insert the menu item in the menu.
MIT_ERROR	An error other than MIT_MEMERROR occurred.
Other	The zero-based index of the offset of the item within the menu.

Remarks

The menu control window procedure responds to this message by inserting the identified item into the menu at the position indicated by the specified *MENUITEM* data structure (contained within the menu-item structure). If the position is MIT_END, the item is added to the end of the menu. If the style of the item includes MIS_TEXT, the text of the item is specified by **itemtext**.

The menu control window procedure sets **indexInserted** to the zero-based index of the position of the item within the menu.

Programming Note: It must be sent, not posted, to the menu control.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set **indexInserted** to the default value of NULL, which is equivalent to zero.

MM_ISITEMVALID

This message returns the selectable status of a specified menu item.

Parameters

param1

item *(USHORT)*
Item identifier.

includesubmenus *(BOOL)*
Include submenus indicator:

TRUE	If the menu does not have an item with the specified identifier, search the submenus and subdialogs of the menu for an item with the specified identifier.
FALSE	If the menu does not have an item with the specified identifier, do not search the submenus and subdialogs of the menu for an item with the specified identifier.

param2 *(BIT32)*
Reserved.

NULL	Reserved value.

Returns

reply

result *(BOOL)*
Selectable indication.

A menu item can be selected and entered under these conditions:

- The item is enabled and, if it is a submenu item, the item in the action bar associated with the submenu is enabled. If the action bar item is not enabled, the user cannot display the submenu.
- The item is enabled, and the submenu is displayed and being tracked with the pointing device or keyboard. It is unlikely, but possible, that the associated action bar is disabled in this instance.

TRUE	The user can select and enter the specified item
FALSE	The user cannot select and enter the specified item.

Remarks

The menu control window procedure responds to this message by setting the return value depending on the selectable status of the specified item.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set **result** to the default value of NULL, which is equivalent to FALSE.

MM_ITEMIDFROMPOSITION

This message returns the identity of a menu item of a specified index.

Parameters

param1

itemIndex *(SHORT)*
Item index.

param2 *(BIT32)*
Reserved.

NULL Reserved value.

Returns

reply

identity *(SHORT)*
Item identity:

MIT_ERROR	Error occurred; for example, because **itemIndex** is not valid.
Other	Item identity.

Remarks

The menu control window procedure responds to this message by setting **reply** to the identity of the item whose position is identified by the index specified in **itemIndex**.

Programming Note: It must be sent, not posted, to the menu control.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set **reply** to the default value of NULL, which is equivalent to zero.

MM_ITEMPOSITIONFROMID

This message returns the index of a menu item of a particular identity.

Parameters

param1

item *(USHORT)*
Item identifier.

includeSubmenus *(BOOL)*
Include submenus indicator:

TRUE	If the menu does not have an item with the specified identifier, search the submenus and subdialogs of the menu for an item with the specified identifier.
FALSE	If the menu does not have an item with the specified identifier, do not search the submenus and subdialogs of the menu for an item with the specified identifier.

param2 *(BIT32)*
Reserved.

NULL Reserved value.

Returns

reply

index *(SHORT)*
Item index:

MIT_NONE	Item does not exist
Other	Item index.

Remarks

The menu control window procedure responds to this message by setting **index** to the zero-based index of the item identified by **item**.

Programming Note: It must be sent, not posted, to the menu control.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set **index** to the default value of NULL, which is equivalent to MIT_NONE.

MM_QUERYITEM

This message returns the definition of the specified menu item.

Parameters

param1

item *(USHORT)*
Item identifier.

includesubmenus *(BOOL)*
Include submenus flag:

TRUE	If the menu does not have an item with the specified identifier, search the submenus and subdialogs of the menu for an item with the specified identifier and copy its definition.
FALSE	If the menu does not have an item with the specified identifier, do not search the submenus and subdialogs of the menu for an item with the specified identifier.

param2

menuItem *(PMENUITEM)*
Menu-item data structure.

This points to a *MENUITEM* structure.

Returns

reply

success *(BOOL)*
Success indicator:

TRUE	Successful completion
FALSE	Error occurred.

Remarks

The menu control window procedure responds to this message by copying the item definition specified by **item**, from the menu, to the structure specified by **menuItem**.

Programming Note: This message does not retrieve the text for items with a style of MIS_TEXT. The item text is obtained by use of the MM_QUERYITEMTEXT message.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set **success** to the default value of NULL, which is equivalent to FALSE.

MM_QUERYITEMATTR

This message returns the attributes of a menu item.

Parameters

param1

item *(USHORT)*
Item identity.

includesubmenus *(BOOL)*
Include submenus indicator:

TRUE If the menu does not have an item with the specified identifier, search the submenus and subdialogs of the menu for an item with the specified identifier and return its state.

FALSE If the menu does not have an item with the specified identifier, do not search the submenus and subdialogs of the menu for an item with the specified identifier.

param2

attributemask *(USHORT)*
Attribute mask.

Returns

reply

state *(USHORT)*
State.

Remarks

The menu control responds to this message by returning the state of the specified attributes of the identified menu item.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set **state** to the default value of NULL, which is equivalent to zero.

MM_QUERYITEMCOUNT

This message returns the number of items in the menu.

Parameters

param1 *(BIT32)*

NULL Reserved value.

param2 *(BIT32)*

NULL Reserved value.

Returns

reply

result *(SHORT)*
Item count.

Remarks

The menu control window procedure responds to this message by returning the count of the number of items in the menu.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set **state** to the default value of NULL, which is equivalent to zero.

MM_QUERYITEMTEXT

This message returns the text of the specified menu item.

Parameters

param1

Item *(USHORT)*
Item identifier.

maxcount *(SHORT)*
Maximum count:

0	Copy all of the item text.
Other	Copy the item text as a null-terminated string, but limit the number of characters copied, including the null termination character, to this value.

param2

Itemtext *(PSTRL)*
Buffer into which the item text is to be copied.

This points to a *STRL*.

Returns

reply

textlength *(SHORT)*
Length of item text.

The length of the text string, excluding the null termination character.

0	Error occurred. For example, no item of the specified identity exists or the item has no text. No text is copied.
Other	Length of item text.

Remarks

The menu control window procedure responds to this message by copying up to **maxcount** characters as a null-terminated string from the text of the item specified by **item**, if it has the style MIS_TEXT, into the buffer specified by **itemtext**.

The length of the item text can be determined by using the MM_QUERYITEMTEXTLENGTH message.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set **textlength** to the default value of NULL, which is equivalent to zero.

MM_QUERYITEMTEXTLENGTH

This message returns the text length of the specified menu item.

Parameters

param1

Item *(USHORT)*
Item identifier.

param2 *(BIT32)*
Reserved.

NULL Reserved value.

Returns

reply

length *(SHORT)*
Length of item text.

The length of the text string, excluding the null termination character.

0 Error occurred. For example, no item of the specified identity exists or the item has no text. No text is copied.

Other Length of item text.

Remarks

The menu control window procedure responds to this message by returning the length in characters of the text of the identified item, if it has a style of MIS_TEXT.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set **length** to the default value of NULL, which is equivalent to zero.

MM_QUERYSELITEMID

This message returns the identity of the selected menu item.

Parameters

param1

reserved *(BIT16)*
Reserved.

NULL

includesubmenus *(BOOL)*
Include submenus indicator:

TRUE If the menu does not have an item with the specified identifier, search the submenus and subdialogs of the menu for a selected item with the specified identifier.

FALSE If the menu does not have an item with the specified identifier, do not search the submenus and subdialogs of the menu for a selected item with the specified identifier.

param2 *(BIT32)*
Reserved.

NULL Reserved value.

Returns

reply

result *(SHORT)*
Selected item identifier:

MID_ERROR	Error occurred
MIT_NONE	No item selected
Other	Selected item identifier.

Remarks

The menu control window procedure responds to this message by returning the identity of the selected item in the menu. Submenus and subdialogs are not searched unless **includesubmenus** is set to TRUE.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set **result** to the default value of NULL, which is equivalent to zero.

MM_REMOVEITEM

This message removes a menu item.

Parameters

param1

item *(USHORT)*
Item identifier.

includesubmenus *(BOOL)*
Include submenus indicator:

TRUE	If the menu does not have an item with the specified identifier, search the submenus and subdialogs of the menu for an item with the specified identifier and delete it.
FALSE	If the menu does not have an item with the specified identifier, do not search the submenus and subdialogs of the menu for an item with the specified identifier.

param2 *(BIT32)*
Reserved.

NULL	Reserved value.

Returns

reply

itemsleft *(SHORT)*
Count of remaining items.

Remarks

The menu control window procedure responds to this message by removing the identified item from the menu and setting **itemsleft** to the count of items in the menu after the item is removed.

The *MENUITEM* structure for the item is not freed by this message; the item may subsequently be inserted into another menu control using the MM_INSERTITEM message.

Programming Note: It must be sent, not posted, to the menu control.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set **itemsleft** to the default value of NULL, which is equivalent to zero.

MM_SELECTITEM

This message selects or deselects a menu item.

Parameters

param1

item *(SHORT)*
Item identifier:

MIT_NONE	Deselect all the items in the menu
Other	Item identifier.

includesubmenus *(BOOL)*
Include submenus indicator:

TRUE	If the menu does not have an item with the specified identifier, search the submenus and subdialogs of the menu for an item with the specified identifier and select or deselect it.
FALSE	If the menu does not have an item with the specified identifier, do not search the submenus and subdialogs of the menu for an item with the specified identifier.

param2

dismissed *(BOOL)*
Dismissed flag:

TRUE	Dismiss the menu
FALSE	Do not dismiss the menu.

Returns

reply

success *(BOOL)*
Success indicator:

TRUE	A selection has been made, or **item** is MIT_NONE
FALSE	A selection has not been made, or a deselection has been made, or **item** is not MIT_NONE.

Remarks

The menu control window procedure responds to this message by setting the selection state of the (sub)menu which contains the specified item to indicate that the item is selected or deselected. If **includesubmenus** is set to TRUE, the selection state of the (sub)menu owning the submenu which contains the specified item is also set. This process continues up the menu hierarchy until the top level menu is reached.

If an item is selected, and **dismissed** is set to TRUE, a WM_COMMAND, WM_SYSCOMMAND, or WM_HELP message, as appropriate, is posted to the owner, and the menu is dismissed.

Programming Note: This message must be sent, not posted, to the menu control.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set **success** to the default value of NULL, which is equivalent to FALSE.

MM_SETITEM

This message sets the definition of a menu item.

Parameters

param1

reserved *(BIT16)*
Reserved.

NULL Reserved value.

includesubmenus *(BOOL)*
Include submenus indicator:

TRUE If the menu does not have an item with the specified identifier, search the submenus and subdialogs of the menu for an item with the specified identifier and set its definition.
FALSE If the menu does not have an item with the specified identifier, do not search the submenus and subdialogs of the menu for an item with the specified identifier.

param2

menuItem *(PMENUITEM)*
Menu-item data structure.

This points to a *MENUITEM* structure.

Returns

reply

success *(BOOL)*
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Remarks

The menu control window procedure responds to this message by using the specified structure to update the definition of the identified menu item.

The **position** field of the structure specified by **menuItem** is ignored, as the position of the item cannot be changed by use of this message.

Programming Note: It must be sent, not posted, to the menu control.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set **success** to the default value of NULL, which is equivalent to FALSE.

MM_SETITEMATTR

This message sets the attributes of a menu item.

Parameters

param1

item *(USHORT)*
Item identifier.

includesubmenus *(BOOL)*
Include submenus indicator:

TRUE If the menu does not have an item with the specified identifier, search the submenus and subdialogs of the menu for an item with the specified identifier and set its attributes.
FALSE If the menu does not have an item with the specified identifier, do not search the submenus and subdialogs of the menu for an item with the specified identifier.

param2

attributemask *(USHORT)*
Attribute mask.

attributedata *(USHORT)*
Attribute data.

Returns

reply

success *(BOOL)*
Success indicator:

TRUE	Successful completion
FALSE	Error occurred.

Remarks

The menu control window procedure responds to this message by setting the state of the specified attributes for the identified item.

Programming Note: It must be sent, not posted, to the menu control.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set **success** to the default value of NULL, which is equivalent to FALSE.

MM_SETITEMHANDLE

This message sets the handle of a menu item.

Parameters

param1

item *(USHORT)*
Item index.

param2

itemhandle *(ULONG)*
Item handle.

Returns

reply

success *(BOOL)*
Success indicator:

TRUE	Successful completion
FALSE	Error occurred.

Remarks

The menu control window procedure responds to this message by setting the handle of the indexed menu item.

This is used to set a handle for menu items that have a style of MIS_BITMAP or MIS_OWNERDRAW.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set **success** to the default value of NULL, which is equivalent to FALSE.

MM_SETITEMTEXT

This message sets the text of a menu item.

Parameters

param1

item *(USHORT)*
Item identifier.

param2

itemtext *(PSTRL)*
Item text.

This points to a *STRL*.

Returns

reply

success *(BOOL)*
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Remarks

The menu control responds to this message by setting the text of the identified item, if it has a style of MIS_TEXT, using the specified null-terminated string.

Programming Note: It must be sent, not posted, to the menu control.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set **success** to the default value of NULL, which is equivalent to FALSE.

MM_STARTMENUMODE

This message is used to begin menu selection.

Parameters

param1

showsubmenu *(BOOL)*
Show submenu flag:

TRUE Show the submenu (pull-down menu) of the selected action bar item when the menu enters selection mode. If the action bar is not visible, the submenu is shown, otherwise it is not shown. If the item selected does not have a submenu, this parameter is ignored.
FALSE Do not show the submenu (pull-down menu) of the selected action bar item when the menu enters selection mode.

resumemenu *(BOOL)*
Resume menu mode flag:

TRUE Resume the user interaction with the menu from where it left off. The menu is assumed to have been used previously and left without dismissing one of the submenus, and therefore is resumed in that submenu.
FALSE Begin user interaction with the menu from the action bar, subject to the value of the **showsubmenu** parameter.

param2 *(BIT32)*
Reserved.

NULL Reserved value.

Returns

reply

success *(BOOL)*
Success indicator:

TRUE	Successful completion
FALSE	Error occurred.

Remarks

It is posted to the menu when the operator presses the menu key.

Programming Note: It must be posted, not sent, to the menu control.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set **success** to the default value of NULL, which is equivalent to FALSE.

WM_QUERYCONVERTPOS

For the cause of this message, see "WM_QUERYCONVERTPOS" on page 12-33.

Parameters

For a description of the parameters, see "WM_QUERYCONVERTPOS" on page 12-33.

Remarks

The menu control window procedure returns QCP_NOCONVERT.

Default Processing

For the default window procedure processing of this message see "WM_QUERYCONVERTPOS" on page 12-33.

WM_QUERYWINDOWPARAMS

Occurs when an application queries the menu control window procedure parameters.

Parameters

For a description of the parameters, see "WM_QUERYWINDOWPARAMS" on page 12-35.

Remarks

The menu control window procedure responds to this message by passing it to the default window procedure.

Default Processing

The default window procedure sets the **length**, **presparamslength**, and **ctldatalength** parameters of the *WNDPARAMS* data structure, identified by **wndparams**, to zero and sets **result** to FALSE.

WM_SETWINDOWPARAMS

This message occurs when an application sets or changes the menu control window procedure parameters.

Parameters

For a description of the parameters, see "WM_SETWINDOWPARAMS" on page 12-40.

Remarks

The menu control window procedure responds to this message by passing it to the default window procedure.

Default Processing

The default window procedure takes no action on this message, other than to set **result** to FALSE.

Chapter 18. Multi-Line Entry Field Control Window Processing

This system-provided window procedure processes the actions on a multi-line entry field control (WC_MLE).

Purpose

An multi-line entry field control is a rectangular window that displays multiple lines of text that the operator can edit. When it has the focus, the cursor marks the current **insertion** or **replacement** point.

How to Use

The text is displayed within a rectangular window. Scroll bars appear if requested.

On all four sides of the text within the window there exists a thin margin area. This margin remains drawn in the window's background color, and characters are never drawn into this margin. Mouse events that occur in the margin are processed differently from mouse events that occur in the text area. The margin should be large enough to be easily clicked-on, but not so large as to take up a large quantity of screen space. It is suggested, but not required, that the left and right margins be half the average character width of the system font, and that the top and bottom margins be half the maximum baseline extent of the system font.

Text is defined as a stream of characters, with hard line-break characters in the text. Between any two bytes in the text stream, and at either end of the document, there is an insertion point. Note that in a DBCS environment, it is possible to have an insertion point in the middle of a DBCS character. If such an insertion point is specified in a function, the function will either round the insertion point in a sensible way, or the function will fail with an error code indicating the problem.

The text always contains a selection region, defined by an anchor point and a cursor point. The anchor and cursor points are insertion points. If the MLE window has the focus, the text between these two points is drawn highlighted and the cursor point is indicated by a flashing text cursor. The selection region can be affected by some import/export operations.

The cursor point and the anchor point define the range of the selection. These two points will often be the same, in which case no text is selected and only a text cursor (but no highlighting) is displayed. A user can use SHIFT+cursor movement combinations to extend the selection, which leaves the anchor point alone, and moves the cursor point to a new position in the document.

The MLE has three modes:

READ-ONLY The keyboard user interface disallows any operations that would change the content of the text, although applications using the MLE can still change the text contents. The application can query this mode, in order that it can disallow application-specific operations.

WORD-WRAP When this mode is in effect, soft line-breaks are inserted into the text at word boundaries so that the user need not scroll the display horizontally to see all the text. When this mode is off, text is allowed to trail off the right-hand edge of the window.

INSERT/OVERTYPE This mode determines whether key strokes are inserted into the text, or whether they overtype existing text. Unlike the other two modes, this mode is maintained by the system. The MLE must merely be aware of the system mode.

Note: In this chapter, 'CR' denotes carriage-return, and 'LF' denotes line-feed.

Multi-Line Entry Field Control Styles

These multi-line entry field control styles are available:

MLS_BORDER A thin border is drawn around the multi-line entry field window.

MLS_READONLY The multi-line entry field is initially in read-only mode.

MLS_WORDWRAP The multi-line entry field initially word-wraps text.

MLS_HSCROLL The multi-line entry field displays and handles a horizontal scroll bar.

MLS_VSCROLL The multi-line entry field displays and handles a vertical scroll bar.

MLS_IGNORETAB The multi-line entry field ignores tab key strokes. It passes the appropriate WM_CHAR to its owner window.

Multi-Line Entry Field Control Data

MLECTLDATA Multi-line entry-field control data structure.

length *(COUNT2B)*
Length of control data in bytes.

format *(USHORT)*
Import/export format.

This sets the initial import/export format. Setting this value via control data is considered identical to setting it through the MLM_FORMAT message. The same constants apply here. The default is MLE_CFTEXT.

textlimit *(COUNT4)*
Text limit.

The maximum amount of text allowed in the MLE. This value is interpreted identically to the parameter of MLM_SETTEXTLIMIT. A negative value indicates that the length is considered unbounded.

anchor *(IPT)*
Selection anchor point.

cursor *(IPT)*
Selection cursor point.

The **anchor** and **cursor** parameters identify the beginning and ending points, respectively, of the selection. These values may range from 0 through the length of the text. The default is 0,0 and can be indicated by entering 0,0.

xformat *(LONG)*
Formatting rectangle width in pixels.

yformat *(LONG)*
Formatting rectangle height in pixels.

The **xformat** and **yformat** parameters identify the dimensions in pixels of the formatting rectangle, as can be set by the MLM_SETFORMATRECT message. These values are considered identical to the two fields in the format rectangle structure referenced in that message, and the interpretation of the values in these fields is governed by the **flags** field.

The default is the window size in both dimensions, and can be indicated by null values.

flags *(ULONG)*
Format flags.

These flags govern the interpretation of the **xformat** and **yformat** fields, just as in the MLM_SETFORMATRECT message. The flag values defined there are also valid in this field. The default is unlimited in both directions and a varying size to match the window size.

Multi-line Entry Field Control Notification Messages

This message is initiated by the multi-line entry field window procedure to notify its owner of significant events.

WM_CONTROL

For the cause of this message, see "WM_CONTROL" on page 12-17.

Parameters

param1

id *(USHORT)*
Control window identity.

notifycode *(USHORT)*
Notify code:

MLN_TEXTOVERFLOW A key stroke causes the amount of text to exceed the limit on the number of bytes of data (refer to MLM_SETTEXTLIMIT). The parameter contains the number of bytes of data which would not fit within the current text limit. For character key strokes this can be 1 or 2 (DBCS). For SHIFT+INS (paste) it can be any amount up to the paste limit.

The default **action** of FALSE causes the default error handling, which is to ignore the key stroke, and beep.

An **action** of TRUE implies that corrective action has been taken (such as deleting existing text or raising the limit) and the WM_CHAR should be reprocessed as if just entered.

MLN_PIXHORZOVERFLOW A key stroke causes the size of the display bit map to exceed the horizontal limit of the format rectangle (refer to MLM_SETFORMATRECT). The parameter contains the number of pels that would not fit within the current text limit.

The default **action** of FALSE causes the default error handling, which is to ignore the key stroke, and beep.

An **action** of TRUE implies that corrective action has been taken (such as changing to a smaller font or raising the limit) and the WM_CHAR should be reprocessed as if just entered.

MLN_PIXVERTOVERFLOW A key stroke causes the size of the display bit map to exceed the vertical limit of the format rectangle (refer to MLM_SETFORMATRECT). The parameter contains the number of pels that would not fit within the current text limit.

The default **action** of FALSE causes the default error handling, which is to ignore the key stroke, and beep.

An **action** of TRUE implies that corrective action has been taken (such as changing to a smaller font or raising the limit) and the WM_CHAR should be reprocessed as if just entered.

MLN_OVERFLOW An action other than entry of a key stroke causes a condition involving the text limit or format rectangle limit, such that either the limit becomes inadequate to contain the text, or the text exceeds the limit.

This can be caused by:
- MLM_SETWRAP
- MLM_SETTABSTOP
- MLM_SETFONT
- MLM_IMPORT
- MLM_PASTE
- MLM_CUT

MLM_UNDO
MLM_DELETE
WM_SIZE.

MLN_HSCROLL Indicates that the MLE has completed a scrolling calculation and is about to update the display accordingly. All queries return values as if the scrolling were complete. However, no scrolling action is visible on the user interface.

MLN_VSCROLL Indicates that the MLE has completed a scrolling calculation and is about to update the display accordingly. All queries return values as if the scrolling were complete. However, no scrolling action is visible on the user interface.

MLN_CHANGE Signals that the text has changed. This notification is sent whenever any text change occurs.

MLN_UNDOOVERFLOW Signals that the text change operation, which could normally be undone, cannot be undone because the amount of text involved exceeds the 'undo' capability. This includes text entry, deletion, cutting, and pasting.

MLN_CLPBDFAIL Signals that a clipboard operation failed.

MLN_MEMERROR Signals that the required storage cannot be obtained. The action that results in the increased storage requirement, fails.

MLN_SETFOCUS Sent whenever the MLE window receives the input focus.

MLN_KILLFOCUS Sent whenever the MLE window loses the input focus.

MLN_MARGIN Whenever the user moves the mouse into the left, right, top, or bottom margins, this message is sent to the window's owner.

If the owner returns an **action** of TRUE, the mouse move is assumed to have been processed by the owner and no further action need be taken.

If the owner returns an **action** of FALSE, the MLE performs a default action appropriate to each different mouse action.

The exceptions to this are all mouse messages that occur after a button-down inside the margin, until and including the matching button-up. Conceptually the 'drag' (button-down until button-up) is a single macro event. Therefore, if FALSE is returned for a button-down event, no further margin notifications are given until after the drag has ended (button-up).

Note: If the application receives a notification of button-down in the margin and processes it, it must capture the mouse until the button-up event.

MLN_SEARCHPAUSE This notification is sent periodically by the MLE, while an MLM_SEARCH message is being processed, to give an application the opportunity to stop excessively long searches, and to provide search progress information. The owner window can respond either with TRUE or FALSE. FALSE causes the MLE to continue searching, and TRUE causes the MLE to stop the search immediately. For further information, see MLM_SEARCH.

param2

This parameter depends on the MLN_* notification code.

For a **notifycode** of MLN_TEXTOVERFLOW:

over *(ULONG)*
Number of bytes that do not fit.

For a **notifycode** of MLN_PIXHORZOVERFLOW or MLN_PIXVERTOVERFLOW:

over *(PIX)*
Linear distance of overflow in pels.

For a **notifycode** of MLN_OVERFLOW:

errinfo *(POVERFLOW)*
Overflow error information structure.

The **errind** field of the *OVERFLOW* structure can take one or more of the following values:

EFR_RESIZE The window is resized, and the format rectangle is tied to the window size and limited either horizontally, vertically, or both. The implicit change of the format rectangle to the new size does not contain the text. The format rectangle is made static at the previous size, and the MLESFR_MATCHWINDOW style is turned off until set again by the application. This is done in response to a WM_SIZE message, and therefore the multi-line entry field does not forward the return value from this notification message.

EFR_TABSTOP A tab stop location change is requested, and the text is limited either horizontally, vertically, or both. Changing the tab stops causes the text to exceed the limit. The tab stop change is rejected.

EFR_FONT A font change is requested and the text is limited either horizontally, vertically, or both. Changing the font causes the text to exceed the limit. The font change is rejected.

EFR_WORDWRAP The word-wrap state is requested to be changed, and the text is limited either horizontally, vertically, or both. Wrapping the text differently exceeds the limit, and the request is rejected. This happens in situations where the horizontal limit is not set, there are lines exceeding it, and word-wrap is being changed from 'off' to 'on' such that it creates soft line-breaks resulting in increased vertical size. This happens if word-wrap is being changed from 'on' to 'off', and there is at least one line created by a soft line-break, such that when that line-break is removed, the full line (up to the hard line-break) exceeds the horizontal limit.

EFR_TEXT Text is changed by MLM_IMPORT, MLM_PASTE, MLM_CUT, MLM_UNDO, or MLM_DELETE, and the text is limited either horizontally, vertically, or both within the format rectangle. The change causes the text to exceed the format rectangle in a dimension that is limited. For example, delete and EOL will join text from two lines into one line long enough to exceed the horizontal limit.

ETL_TEXTBYTES Text is changed by MLM_IMPORT, MLM_PASTE, or MLM_UNDO, and the text is limited to a maximum number of bytes. The change causes the text to exceed that maximum.

For a **notifycode** of MLN_CLPBDFAIL:

errind *(ULONG)*
Clipboard fail flag.

For a **notifycode** of MLN_MARGIN:

EFR_TOOMUCHTEXT Text amount exceeds clipboard capacity
EFR_CLPBDERROR A clipboard error occurred.

mrg *(PMARGSTRUCT)*
Margin structure.

The left and right margins are defined as going all the way to the top and bottom such that the top and bottom margins are contained between them. Therefore, the corners are included in the sides.

moumsg contains the mouse message that signals the event.

near contains the insertion point of the nearest point in the text. For situations where the nearest location is beyond the end of a line, the insertion point for the end of the line is returned. (The EOL character is considered to be beyond the end of the line.)

For a **notifycode** of MLN_SEARCHPAUSE:

searchedto *(IPT)*
Current insertion point of search.

For a **notifycode** of MLN_HSCROLL, MLN_VSCROLL, MLN_CHANGE, MLN_UNDOOVERFLOW, MLN_MEMERROR, MLN_SETFOCUS, or MLN_KILLFOCUS:

reserved *(BIT32)*
Reserved.

NULL

Returns

reply

For a **notifycode** of MLN_TEXTOVERFLOW, MLN_PIXHORZOVERFLOW, MLN_PIXVERTOVERFLOW, MLN_MARGIN, or MLN_SEARCHPAUSE:

action *(BOOL)*
Action taken by application:

For a **notifycode** of MLN_OVERFLOW, MLN_HSCROLL, MLN_VSCROLL, MLN_CHANGE, MLN_UNDOOVERFLOW, MLN_CLPBDFAIL, MLN_MEMERROR, MLN_SETFOCUS, or MLN_KILLFOCUS:

TRUE The multi-line entry field control assumes that appropriate action has been taken by the application. Appropriate action depends on the MLN_* notification code, and is documented under the **notifycode** field.
FALSE The multi-line entry field control assumes that the application has ignored this WM_CONTROL message, and takes action appropriate to the MLN_* notification code, as documented under the **notifycode** field.

reserved *(BIT32)*
Reserved.

NULL Reserved null.

Remarks

The multi-line entry field control window procedure generates this message and sends it to its owner, informing the owner of the event.

Default Processing

The default window procedure takes no action on this message, other than to set **reply** to NULL.

Multi-line Entry Field Window Messages

This section describes the multi-line entry field control window procedure actions on receiving the following messages.

MLM_CLEAR

This message clears the current selection.

Parameters

param1 *(BIT32)*
Reserved.

NULL Reserved value.

param2 *(BIT32)*
Reserved.

NULL Reserved value.

Returns

reply

clear *(ULONG)*
Number of bytes deleted, counted in CF_TEXT format.

Remarks

The multi-line entry field control window procedure responds to this message by clearing the current selection and returning the number of bytes cleared.

Default Processing

The default window procedure takes no action on this message, other than to set **reply** to NULL.

MLM_COPY

This message copies the current selection to the clipboard.

Parameters

param1 *(BIT32)*
Reserved.

NULL Reserved value.

param2 *(BIT32)*
Reserved.

NULL Reserved value.

Returns

reply

copy *(ULONG)*
Number of bytes transferred, counted in CF_TEXT format.

Remarks

The multi-line entry field control window procedure responds to this message by copying the selected text to the clipboard. The text is translated to standard clipboard format, which is the same as exporting with MLE_CFTEXT format.

The text is placed on the clipboard as a single contiguous data segment. This restricts the amount to the maximum segment size (64KB).

This may cause an overflow, see MLN_OVERFLOW.

Default Processing

The default window procedure takes no action on this message, other than to set **reply** to NULL.

MLM_CUT

This message copies the text that forms the current selection to the clipboard and then deletes it from the MLE control.

Parameters

param1 *(BIT32)*
Reserved.

NULL Reserved value.

param2 *(BIT32)*
Reserved.

NULL Reserved value.

Returns

reply

copy *(ULONG)*
Number of bytes transferred, counted in CF_TEXT format.

Remarks

The multi-line entry field control window procedure responds to this message by copying the selected text to the clipboard and then deleting it. The text is translated to standard clipboard format, which is the same as exporting with MLE_CFTEXT format.

The text is placed on the clipboard as a single contiguous data segment. This restricts the amount to the maximum segment size (64KB).

This may cause an overflow, see MLN_OVERFLOW.

Default Processing

The default window procedure takes no action on this message, other than to set **reply** to NULL.

MLM_CHARFROMLINE

This message returns the first insertion point on a given line.

Parameters

param1

linenum *(LONG)*
Line number of interest.

param2 *(BIT32)*
Reserved.

NULL Reserved value.

Returns

reply

first *(IPT)*
First insertion point on line.

Remarks

For any line number, the insertion point just before the first character on that line is returned. If the line number is −1, the line containing the cursor is used.

The term 'line' means a line on the display after the application of word-wrap. It does not mean a line as defined by the CR LF line-break sequence.

Default Processing

The default window procedure takes no action on this message, other than to set **reply** to NULL.

MLM_DELETE

This message deletes text.

Parameters

param1

begin *(IPT)*
Starting point of deletion.

param2

del *(ULONG)*
Number of bytes to delete.

Returns

reply

success *(ULONG)*
Number of bytes successfully deleted.

Remarks

This message takes an insertion point and a length, and deletes that number of characters from the text. If the insertion point is −1, the selection is used and the effect is identical to the MLM_CLEAR message.

This may cause an overflow, see MLN_OVERFLOW.

Default Processing

The default window procedure takes no action on this message, other than to set **reply** to NULL.

MLM_DISABLEREFRESH

This message disables screen refresh.

Parameters

param1 *(BIT32)*
Reserved.

NULL Reserved value.

param2 *(BIT32)*
Reserved.

NULL Reserved value.

Returns

reply

success *(BOOL)*
Success indicator:

TRUE Successful completion
FALSE An error occurred.

Remarks

This message disables screen refreshes. This allows an application to make changes throughout a document while avoiding unnecessary overhead caused by attempts to keep the screen display current. When an MLM_ENABLEREFRESH message is sent, the screen display is brought up to date with the contents of the text.

While refresh is disabled, mouse and keyboard messages are processed by beeping and ignoring them, except for mouse moves which do not beep; the mouse pointer changes to the system standard 'wait' symbol (an hourglass).

Default Processing

The default window procedure takes no action on this message, other than to set **reply** to NULL.

MLM_ENABLEREFRESH

This message enables screen refresh.

Parameters

param1 *(BIT32)*
Reserved.

NULL Reserved value.

param2 *(BIT32)*
Reserved.

NULL Reserved value.

Returns

reply

success *(BOOL)*
Success indicator:

TRUE Successful completion
FALSE An error occurred.

Remarks

This message disables screen refreshes. This allows an application to make changes throughout a document while avoiding unnecessary overhead caused by attempts to keep the screen display current. When an MLM_ENABLEREFRESH message is sent, the screen display is brought up to date with the contents of the text.

While refresh is disabled, mouse and keyboard messages are processed by beeping and ignoring them, except for mouse moves which do not beep; the mouse pointer changes to the system standard 'wait' symbol (an hourglass).

Default Processing

The default window procedure takes no action on this message, other than to set **reply** to NULL.

MLM_EXPORT

This message exports text to a buffer.

Parameters

param1

begin *(PIPT)*
Starting point.

Updated to follow the last character exported.

param2

copy *(PULONG)*
Number of bytes being exported.

Decremented by the number of bytes actually exported.

Returns

reply

success *(ULONG)*
Number of bytes successfully exported.

Remarks

Takes an insertion point and length as parameters. Copies text starting from that insertion point into the buffer set by MLM_SETIMPORTEXPORT. Text will be in the format set by MLM_FORMAT. If the insertion point is −1, the selection will be used for both **begin** and **copy**.

On return **begin** is updated to follow the last byte exported and the number of bytes to be exported is decremented the number actually exported. This is done to prepare those parameter values for the next export. The return value indicates the number of bytes actually put into the buffer. This number will be less than or equal to the buffer size (see MLM_SETIMPORTEXPORT).

Note: All exports are done in full characters. Therefore if either the length of the buffer or the number of bytes to be exported result in the last byte transferred being only half of a DBCS character, the MLE will ***not*** transfer that byte.

Returns the number of bytes placed in the export buffer.

Default Processing

The default window procedure takes no action on this message, other than to set **reply** to NULL.

MLM_FORMAT

This message sets the format to be used for buffer importing and exporting.

Parameters

param1

format *(USHORT)*
Format to be used for import/export.

MLE_CFTEXT	Clipboard text format, uses CR LF for line delineation on export, and recognizes either LF, CR LF, or LF CR as a line-break on import.
MLE_NOTRANS	Uses LF for line delineation, and guarantees that any text imported into the MLE in this format can be recovered in exactly the same form on export.
MLE_WINFMT	(Windows MLE format.) On import, recognizes CR LF as denoting hard line-breaks, and ignores the sequence CR CR LF. On export, uses CR LF to denote a hard line-break and CR CR LF to denote a soft line-break caused by word-wrapping.

param2 *(BIT32)*
Reserved.

NULL Reserved value.

Returns

reply

format *(USHORT)*
Previous format value.

Remarks

The default format is MLE_CFTEXT.

The keyword MLE_RTF is reserved.

Default Processing

The default window procedure takes no action on this message, other than to set **reply** to NULL.

MLM_IMPORT

This message imports text from a buffer.

Parameters

param1

begin *(PIPT)*
Insertion point. Updated to insertion point following last insert.

param2

copy *(ULONG)*
Number of bytes in buffer.

Returns

reply

success *(ULONG)*
Number of bytes successfully inserted.

Remarks

This message takes an insertion point and length as parameters. It assumes a buffer has been set using MLM_SETIMPORTEXPORT, and inserts the contents of the buffer at the insertion point in the text. The contents are interpreted as being in the format set by MLM_FORMAT. If the insertion point is −1, the cursor point is used.

The insertion point **begin** is updated by the MLE to the point after the last character imported. This provides the application with the location for the next import.

The return value indicates how many bytes were actually transferred.

All imports are done in full characters, therefore, if the number of bytes to be imported results in the last byte transferred being only half of a DBCS character, or part of a line-break sequence (CR LF or CR CR LF), the MLE does *not* transfer that byte. If the return value indicates that less than the full amount was transferred, a check must be made to determine if it is the beginning of a multi-byte sequence, and if so, the parts must be mated and imported as a whole.

This can cause an overflow, see MLN_OVERFLOW.

Note: The buffer is not zero-terminated; NULL characters can be inserted into the text.

Default Processing

The default window procedure takes no action on this message, other than to set **reply** to NULL.

MLM_INSERT

This message deletes the current selection and replaces it with a text string.

Parameters

param1

text *(STRL)*
Null-terminated text string.

param2 *(BIT32)*
Reserved.

NULL Reserved value.

Returns

reply

count *(ULONG)*
Number of bytes actually inserted.

Remarks

This message inserts the text string at the current selection, deleting that selection in the same manner as typing at the keyboard would. The text string must be in CF_TEXT format (or one of the formats acceptable to MLM_IMPORT) and null-terminated. The line-break (CR LF, LF, etc.) is counted as one byte, regardless of the number of bytes occupied in the buffer, and the null terminator is not counted.

This interacts with the format rectangle and text limits, and a return of less than the full count can be the result. If so, a notification message is sent.

Default Processing

The default window procedure takes no action on this message, other than to set **reply** to NULL.

MLM_LINEFROMCHAR

This message returns the line number corresponding to a given insertion point.

Parameters

param1

first *(IPT)*
Insertion point of interest

param2 *(BIT32)*
Reserved.

NULL Reserved value.

Returns

reply

linenum *(LONG)*
Line number of insertion point.

Remarks

For any insertion point, the corresponding line number is returned. If the insertion point is −1, the number of the line containing the first insertion point of the selection is returned.

The term 'line' means a line on the display after the application of word-wrap. It does not mean a line as defined by the CR LF line-break sequence.

Default Processing

The default window procedure takes no action on this message, other than to set **reply** to NULL.

MLM_PASTE

This message replaces the text that forms the current selection, with text from the clipboard.

Parameters

param1 *(BIT32)*
Reserved.

NULL Reserved value.

param2 *(BIT32)*
Reserved.

NULL Reserved value.

Returns

reply

copy *(ULONG)*
Number of bytes transferred, counted in CF_TEXT format.

Remarks

The multi-line entry field control window procedure responds to this message by replacing the selected text with text from the clipboard. The text is translated from standard clipboard format, which is the same as importing with MLE_CFTEXT format.

The text is assumed to be in the clipboard as a single contiguous data segment. This restricts the amount to the maximum segment size (64KB).

This can cause an overflow, see MLN_OVERFLOW.

Default Processing

The default window procedure takes no action on this message, other than to set **reply** to NULL.

MLM_QUERYBACKCOLOR

This message queries the background color.

Parameters

param1 *(BIT32)*
Reserved.

NULL Reserved value.

param2 *(BIT32)*
Reserved.

NULL Reserved value.

Returns

reply

color *(LONG)*
Text color.

Remarks

This message returns the color in which the background is to be drawn.

The color values are the same as those used by GpiSetColor.

Default Processing

The default window procedure takes no action on this message, other than to set **reply** to NULL.

MLM_QUERYCHANGED

This message queries the changed flag.

Parameters

param1 *(BIT32)*
Reserved.

NULL Reserved value.

param2 *(BIT32)*
Reserved.

NULL Reserved value.

Returns

reply

changed *(BOOL)*
Current changed status.

TRUE Text has changed since the last time that the change flag was cleared
FALSE Text has not changed since the last time that the change flag was cleared.

Remarks

The multi-line entry field control window procedure responds to this message by returning the changed flag for the text without altering it. See also MLN_CHANGE.

Default Processing

The default window procedure takes no action on this message, other than to set **reply** to NULL.

MLM_QUERYFIRSTCHAR

This message queries the first visible character.

Parameters

param1 *(BIT32)*
Reserved.

NULL Reserved value.

param2 *(BIT32)*
Reserved.

NULL Reserved value.

Returns

reply

fvc *(IPT)*
First visible character.

Remarks

Returns the insertion point immediately preceding the character visible in the upper left-hand corner of the screen. If a partial character is displayed, that character counts as the first visible character.

Note: In situations where no character is visible, because the text is scrolled to the right beyond the end of the top line, this returns the insertion point of the last character on the line (EOL not considered). In situations where there are no characters on the line, the insertion point at the beginning is returned.

Default Processing

The default window procedure takes no action on this message, other than to set **reply** to NULL.

MLM_QUERYFONT

This message queries which font is in use.

Parameters

param1

fattrs *(PFATTRS)*
Font attribute structure.

param2 *(BIT32)*
Reserved.

NULL Reserved value.

Returns

reply

system *(BOOL)*
System font indicator:

TRUE The system font is in use
FALSE The system font is not in use.

Remarks

This message puts the attributes of the current drawing font, into the font attributes structure.

Default Processing

The default window procedure takes no action on this message, other than to set **reply** to NULL.

MLM_QUERYFORMATLINELENGTH

This message returns the number of bytes to end of line after formatting has been applied.

Parameters

param1

start *(IPT)*
Insertion point to count from.

param2 *(BIT32)*
Reserved.

NULL Reserved value.

Returns

reply

line *(IPT)*
Count of bytes to end of line.

Remarks

For any insertion point, the number of bytes between that insertion point and the end of the line, is returned, after the current formatting is applied. If the insertion point is −1, the cursor position is used. This message differs from MLM_QUERYLINELENGTH in that the byte count returned reflects the effects of the current formatting set by MLM_FORMAT.

Default Processing

The default window procedure takes no action on this message, other than to set **reply** to NULL.

MLM_QUERYFORMATTEXTLENGTH

This message returns the length of a specified range of characters after the current formatting has been applied.

Parameters

param1

start *(IPT)*
Insertion point to start from.

param2

scan *(ULONG)*
Number of characters to convert to bytes.

0xFFFFFFFF	Convert until end of line
other	Convert specified number of characters.

Returns

reply

text *(ULONG)*
Count of bytes in text after formatting.

Remarks

This message returns the length in bytes of a range of characters after the current formatting is applied. This differs from MLM_QUERYTEXTLENGTH in that:

- A range of insertion points can be queried.
- The byte count returned reflects the effects of the current formatting set by MLM_FORMAT.

Default Processing

The default window procedure takes no action on this message, other than to set **reply** to NULL.

MLM_QUERYFORMATRECT

This message queries the format dimensions and mode.

Parameters

param1

formatrect *(PPOINTL)*
Format dimensions.

The size of the current limiting dimensions.

param2

flags *(BIT32)*
Flags governing interpretation of dimensions.

An array of MLESFR_* flags defined under the **flags** field of the MLM_SETFORMATRECT message.

Returns

reply *(BIT32)*
Reserved.

Default Processing

The default window procedure takes no action on this message, other than to set **reply** to NULL.

MLM_QUERYIMPORTEXPORT

This message queries the current transfer buffer.

Parameters

param1

buff *(PPSTRL)*
Transfer buffer.

param2

buff *(PULONG)*
Size of transfer buffer in bytes.

Returns

reply

count *(ULONG)*
Success indicator:

TRUE	Successful completion
FALSE	An error occurred.

Remarks

This message returns the values from the most recent MLM_SETIMPORTEXPORT, or zero for either value if it has not been set.

Default Processing

The default window procedure takes no action on this message, other than to set **reply** to NULL.

MLM_QUERYLINECOUNT

This message queries the number of lines of text.

Parameters

param1 *(BIT32)*
Reserved.

NULL Reserved value.

param2 *(BIT32)*
Reserved.

NULL Reserved value.

Returns

reply

lines *(ULONG)*
The number of lines of text.

Remarks

The term 'line' means a line on the display after the application of word-wrap. It does not mean a line as defined by the CR LF line-break sequence.

Default Processing

The default window procedure takes no action on this message, other than to set **reply** to NULL.

MLM_QUERYLINELENGTH

This message returns the number of bytes between a given insertion point and the end of line

Parameters

param1

start *(IPT)*
Insertion point to count from.

param2 *(BIT32)*
Reserved.

NULL Reserved value.

Returns

reply

line *(IPT)*
Count of bytes to end of line.

Remarks

For any insertion point, the number of bytes between that insertion point and the end of the line is returned. If the insertion point is −1, the cursor position is used. If the line contains a hard line-break, it is counted as one byte.

The term 'line' means a line on the display after the application of word-wrap. It does not mean a line as defined by the CR LF line-break sequence.

Default Processing

The default window procedure takes no action on this message, other than to set **reply** to NULL.

MLM_QUERYREADONLY

This message queries the read-only mode.

Parameters

param1 *(BIT32)*
Reserved.

NULL Reserved value.

param2 *(BIT32)*
Reserved.

NULL Reserved value.

Returns

reply

readonly *(BOOL)*
Current read-only status.

TRUE Read-only mode is set
FALSE Read-only mode is cleared.

Default Processing

The default window procedure takes no action on this message, other than to set **reply** to NULL.

MLM_QUERYSEL

This message returns the location of the selection.

Parameters

param1

querymode *(USHORT)*
Query Mode.

0 Return both minimum and maximum points of selection in a format compatible with the EM_QUERYSEL message.
1 Return minimum insertion point of selection.
2 Return maximum insertion point of selection.
3 Return anchor point of selection.
4 Return cursor point of selection.

param2 *(BIT32)*
Reserved.

NULL Reserved value.

Returns

reply

For **querymode** = 0:

minsel *(SHORT)*
Minimum insertion point of selection.

This value is rounded down to 65 535, if necessary.

maxsel *(SHORT)*
Maximum insertion point of selection.

This value is rounded down to 65 535, if necessary.

For **querymode** = 1, 2, 3, or 4:

ipt *(IPT)*
Requested insertion point.

Remarks

This message returns the location of the selection in several different forms. The insertion points lie between characters, and start at a zero origin before the first character in the MLE. Subtracting the minimum from the maximum gives the number of characters in the selection. *This is not necessarily the number of bytes of ASCII.* The line-break character is a CR LF (2 bytes) and all DBCS characters are two bytes. To determine the number of bytes, use MLM_QUERYFORMATTEXTLENGTH, being sure that the format choice set by MLM_FORMAT is set to what is used when the data is exported from the MLE (for example, MLE_CFTEXT for MLM_QUERYSELTEXT).

Note the following:

- If anchor point > cursor point, minimum point = cursor point and maximum point = anchor point.
- If anchor point < cursor point, minimum point = anchor point and maximum point = cursor point.

Default Processing

The default window procedure takes no action on this message, other than to set **reply** to NULL.

MLM_QUERYSELTEXT

This message copies the currently selected text into a buffer.

Parameters

param1

buff *(STRL)*
Buffer for text string.

param2 *(BIT32)*
Reserved.

NULL Reserved value.

Returns

reply

count *(ULONG)*
Number of bytes to put into text string.

Remarks

This message copies the currently selected text into the buffer pointed to by **buff**. The text string is null-terminated. The byte count includes the text in CF_TEXT format (CR LF) and the null terminator.

Default Processing

The default window procedure takes no action on this message, other than to set **reply** to NULL.

MLM_QUERYTABSTOP

This message queries the pel interval at which tab stops are placed.

Parameters

param1 *(BIT32)*
Reserved.

NULL Reserved value.

param2 *(BIT32)*
Reserved.

NULL Reserved value.

Returns

reply

tabset *(PIX)*
Tab width in pels.

<0 An error occurred.
Other The pel interval at which tab stops are placed.

Remarks

This message fails and returns a negative value, if the reserved values are not zero.

Default Processing

The default window procedure takes no action on this message, other than to set **reply** to NULL.

MLM_QUERYTEXTCOLOR

This message queries the text color.

Parameters

param1 *(BIT32)*
Reserved.

NULL Reserved value.

param2 *(BIT32)*
Reserved.

NULL Reserved value.

Returns

reply

color *(LONG)*
Text color.

Remarks

This message returns the color in which text is to be drawn.

The color values are the same as those used by GpiSetColor.

Default Processing

The default window procedure takes no action on this message, other than to set **reply** to NULL.

MLM_QUERYTEXTLENGTH

This message returns the number of characters in the text.

Parameters

param1 *(BIT32)*
Reserved.

NULL Reserved value.

param2 *(BIT32)*
Reserved.

NULL Reserved value.

Returns

reply

text *(IPT)*
Count of text in bytes.

Remarks

This message returns the number of characters in the text. Hard line-breaks are counted as 1 and soft-line breaks as 0.

This message differs from the WinQueryWindowTextLength call in that it returns a *LONG*.

Default Processing

The default window procedure takes no action on this message, other than to set **reply** to NULL.

MLM_QUERYTEXTLIMIT

This message queries the maximum number of bytes that a multi-line entry field control can contain.

Parameters

param1 *(BIT32)*
Reserved.

NULL Reserved value.

param2 *(BIT32)*
Reserved.

NULL Reserved value.

Returns

reply

size *(LONG)*
Maximum number of bytes allowed in the MLE.

Remarks

The multi-line entry field control window procedure responds to this message by returning the current limit set, either by default, or by MLM_SETTEXTLIMIT. If the limit is unbounded, a non-positive value will be returned.

Default Processing

The default window procedure takes no action on this message, other than to set **reply** to NULL.

MLM_QUERYUNDO

This message queries the 'undo' or 'redo' operations that are possible.

Parameters

param1 *(BIT32)*
Reserved.

NULL Reserved value.

param2 *(BIT32)*
Reserved.

NULL Reserved value.

Returns

reply

operation *(USHORT)*
Operation that can be undone or redone.

0	An 'undo' or 'redo' operation is not possible.
WM_CHAR	A WM_CHAR message, or messages for a simple string of key strokes, can be undone or redone.
MLM_SETFONT	A MLM_SETFONT message can be undone or redone.
MLM_SETTEXTCOLOR	A MLM_SETTEXTCOLOR message can be undone or redone for both background and foreground color.
MLM_CUT	A MLM_CUT message can be undone or redone.
MLM_PASTE	A MLM_PASTE message can be undone or redone.
MLM_CLEAR	A MLM_CLEAR message can be undone or redone.

undoredo *(BOOL)*
'Undo' or 'redo' indicator.

TRUE	An 'undo' is possible
FALSE	A 'redo' is possible.

Default Processing
The default window procedure takes no action on this message, other than to set **reply** to NULL.

MLM_QUERYWRAP
This message queries the wrap flag.

Parameters
param1 *(BIT32)*
Reserved.

NULL	Reserved value.

param2 *(BIT32)*
Reserved.

NULL	Reserved value.

Returns
reply

wrap *(BOOL)*
Wrap flag.

TRUE	Word-wrap enabled
FALSE	Word-wrap disabled.

Default Processing
The default window procedure takes no action on this message, other than to set **reply** to NULL.

MLM_RESETUNDO
This message resets the 'undo' state to indicate that no 'undo' operations are possible.

Parameters
param1 *(BIT32)*
Reserved.

NULL	Reserved value.

param2 *(BIT32)*
Reserved.

NULL	Reserved value.

Returns
reply

operation *(USHORT)*
Operation which can be undone or redone.

0	An 'undo' or 'redo' operation is not possible.
WM_CHAR	A WM_CHAR message or messages for a simple string of key strokes can be undone or redone.
MLM_SETFONT	A MLM_SETFONT message can be undone or redone.
MLM_SETTEXTCOLOR	A MLM_SETTEXTCOLOR message can be undone or redone for both background and foreground color.
MLM_CUT	A MLM_CUT message can be undone or redone.
MLM_PASTE	A MLM_PASTE message can be undone or redone.
MLM_CLEAR	A MLM_CLEAR message can be undone or redone.

undoredo *(BOOL)*
'Undo' or 'redo' indicator.

TRUE	An 'undo' is possible
FALSE	A 'redo' is possible.

Remarks

This message resets the 'undo' state of the MLE to indicate that the last operation cannot be undone (null return from MLM_QUERYUNDO). This can be used by the application when it performs an operation that it can undo, that supersedes the last MLE operation. The application can then reset its own 'undo' state upon receipt of an MLN_CHANGE, indicating that later changes have occurred through the MLE.

Default Processing

The default window procedure takes no action on this message, other than to set **reply** to NULL.

MLM_SEARCH

This message searches for a specified text string.

Parameters

param1

style *(ULONG)*
Style flags.

MLSS_CASESENSITIVE	If set, only exact matches are considered a successful match. If not set, any case-combination of the correct characters in the correct sequence is considered a successful match.
MLSS_SELECTMATCH	If set, the MLE selects the text and scrolls it into view when found, just as if the application had sent an MLM_SETSEL message. This is not done if MLSS_CHANGEALL is also indicated.
MLSS_CHANGEALL	Using the *SEARCH* structure specified in **se**, all occurrences of **findstring** are found, searching from **start** to **stop**, and replacing them with **changestring**. If this style is selected, the **founsstringlength** field has no meaning and, the **start** value points to the place where the search stopped, or is the same as **stop** because the search has not stopped at any of the found strings. The current cursor location is not moved, however, any existing selection is deselected.

param2

se *(PMLE_SEARCHDATA)*
Search specification structure.

Returns

reply

success *(BOOL)*
Success indicator:

TRUE	The search was successful
FALSE	The search was unsuccessful.

Remarks

This message searches the MLE text for a specified string, starting at a specified insertion point and continuing until the second specified insertion point has been reached, or the requested string has been matched.

When an MLM_SEARCH message is sent, the text is scanned starting with the character that follows the insertion point indicated in the **start** field of the *SEARCH* structure. The search proceeds until the point indicated in the **stop** field, until a match is found, or until TRUE is returned from

MLN_SEARCHPAUSE notification (see WM_CONTROL). If a negative value is specified for the **start**, the current cursor point is used. If a negative value is specified for **stop**, the end of the text is used. If **stop**, is less than or equal to **start**, after performing the two indicated substitutions, the search wraps from the end of the text to the beginning of the text.

If the MLSS_CASESENSITIVE option is specified, the bytes of the search string must exactly match those in the text. If MLS_CASESENSITIVE is not specified, the WinUpperChar of the search string must match the WinUpperChar of the text.

When a match is found, the **start** field of the search specification structure is set to indicate the insertion point immediately preceding the first character of the match, and the **findstringlength** field is set to indicate the number of characters in the match. The cursor/selection is not altered unless MLSS_SELECTMATCH is specified. If it is, an MLM_SETSEL is done with the anchor point at **start** and the cursor at **start** + **findstringlength**.

While searching, the MLE occasionally sends an MLN_SEARCHPAUSE notification message. If the owner responds to this message with the value TRUE, the MLE stops the search. When a search is stopped from MLN_SEARCHPAUSE, **start** is set to the point where the search terminated. If the response is FALSE, the search continues, (see also the definition of MLN_SEARCHPAUSE). The interval at which MLN_SEARCHPAUSE notifications are sent is implementation-dependent, but must not exceed reasonable user-response thresholds, nor should it be so often as to introduce undue messaging overhead. Sending this notification every half second is a reasonable compromise.

When no match is found the **start** value is unchanged.

If the application wishes to continue the search, the proper way is to change the **start** value to be the point following the string found, adjusting for any text changes done after the search that may have moved the relative location of the point.

Applications using this message are advised to change the system pointer to the 'wait' icon (the hourglass) if it is expected that the search will take some time.

Default Processing

The default window procedure takes no action on this message, other than to set **reply** to NULL.

MLM_SETBACKCOLOR

This message sets the background color.

Parameters

param1

color *(LONG)*
Color.

param2 *(BIT32)*
Reserved.

NULL Reserved value.

Returns

reply

oldcolor *(LONG)*
Color previously used.

Remarks

This message sets the color in which the MLE background is to be drawn, and updates the display as necessary.

The color values are the same as those used by GpiSetColor.

Default Processing

The default window procedure takes no action on this message, other than to set **reply** to NULL.

MLM_SETCHANGED

This message sets or clears the changed flag.

Parameters

param1

changednew *(BOOL)*
Value to set changed flag to.

param2 *(BIT32)*
Reserved.

NULL Reserved value.

Returns

reply

changed *(BOOL)*
Changed status before message was processed.

TRUE Text has changed since the last time that the change flag was cleared
FALSE Text has not changed since the last time that the change flag was cleared.

Remarks

This message can generate a MLN_CHANGE notification.

Default Processing

The default window procedure takes no action on this message, other than to set **reply** to NULL.

MLM_SETFIRSTCHAR

This message sets the first visible character.

Parameters

param1

fvc *(IPT)*
Insertion point to place in top left-hand corner.

param2 *(BIT32)*
Reserved.

NULL Reserved value.

Returns

reply

success *(BOOL)*
Success indicator:

TRUE Successful completion
FALSE An error occurred.

Remarks

This message scrolls the text to place the character following the insertion point into the upper left-hand corner of the window. If the insertion point specified is beyond the end of a line, or the end of the file, it is resolved in the same way as it is for a mouse 'click'.

Default Processing

The default window procedure takes no action on this message, other than to set **reply** to NULL.

MLM_SETFONT

This message sets a font.

Parameters

param1

fattrs *(PFATTRS)*
Font attribute structure.

NULL The system font is set
other The specified font is set.

param2 *(BIT32)*
Reserved.

NULL Reserved value.

Returns

reply

success *(BOOL)*
Success indicator:

TRUE The font was successfully set
FALSE An error occurred.

Remarks

For any *PFATTRS*, this message sets the display to use the appropriate font. If NULL, the system font is used. The screen is updated appropriately.

This can cause an overflow, see MLN_OVERFLOW.

Default Processing

The default window procedure takes no action on this message, other than to set **reply** to NULL.

MLM_SETFORMATRECT

This message sets the format dimensions and mode.

Parameters

param1

formatrect *(PPOINTL)*
New format dimensions.

NULL A null value sets both dimensions to the current window size.
other The structure is a pair of *LONG*s designating the diagonally-opposite corner of the rectangle assuming, 0,0 for the first. Therefore, they are the width and height in pels of the format rectangle. These dimensions are used as the word-wrap and text-size limiting boundaries. Negative values for either dimension cause the MLE to substitute the current window size (the MLE window rectangle minus margins).

If the rectangle specified has either, or both, of the limits set, and the size is inadequate to contain the text, **success** is set to FALSE and the rectangle dimensions are replaced with the overflow amounts.

param2

flags *(BIT32)*
Flags governing interpretation of dimensions.

MLESFR_MATCHWINDOW	The dimensions of the format rectangle are always to be kept the same as the window size minus the margins. This causes the MLE, implicitly, to do a MLM_SETFORMATRECT each time the window is resized, and effectively causes any other dimensions to be ignored. Resizing of the window can cause this setting to be automatically negated (see MLN_OVERFLOW).
MLESFR_LIMITHORZ	The width of any line in the MLE cannot exceed the given horizontal dimension. If word-wrap is on, this limit has no effect. Word-wrap can result in trailing blanks beyond the right limit. These do not cause an overflow notification.
MLESFR_LIMITVERT	The vertical height of the total text, as displayed, is limited to that which fits totally within the vertical dimension of the format rectangle.

Returns

reply

success *(BOOL)*
Success indicator:

TRUE	Successful completion
FALSE	An error occurred.

Remarks

The multi-line entry field control window procedure responds to this message by setting formatting dimensions and mode.

Any addition of text that causes the text to exceed the rectangle limits causes a notification prior to proceeding (see MLN_PIXHORZOVERFLOW and MLN_PIXVERTOVERFLOW).

Any activity that would cause the rectangle to be unable to contain the existing text (resize, undo, increasing font size, or word-wrap on or off) is rejected and results in a notification message for information (see MLN_OVERFLOW).

Default Processing

The default window procedure takes no action on this message, other than to set **reply** to NULL.

MLM_SETIMPORTEXPORT

This message sets the current transfer buffer.

Parameters

param1

buff *(PSTRL)*
Transfer buffer.

param2

length *(ULONG)*
Size of transfer buffer in bytes.

Returns

reply

success *(BOOL)*
Success indicator:

TRUE	Successful completion
FALSE	An error occurred.

Remarks

Given a far pointer to a buffer, and the size of the buffer, this message sets it as the current transfer buffer for the MLE. This buffer is used by the MLM_IMPORT and MLM_EXPORT messages. The system segment limit must be observed when specifying the buffer size.

Default Processing

The default window procedure takes no action on this message, other than to set **reply** to NULL.

MLM_SETSEL

This message sets a selection.

Parameters

param1

anchor *(IPT)*
Insertion point for new anchor point.

param2

cursor *(IPT)*
Insertion point for new cursor point.

Returns

reply

success *(BOOL)*
Success indicator:

TRUE	Successful completion
FALSE	An error occurred.

Remarks

This message sets the anchor and cursor points. The screen display is updated appropriately, ensuring that the cursor point is visible (which may involve scrolling). Note that the text cursor and inversion are not displayed if the MLE window does not have the input focus. A negative value for a point leaves that point alone.

Default Processing

The default window procedure takes no action on this message, other than to set **reply** to NULL.

MLM_SETREADONLY

This message sets or clears read-only mode.

Parameters

param1

readonly *(BOOL)*
New read-only value.

param2 *(BIT32)*
Reserved.

NULL	Reserved value.

Returns

reply

old *(BOOL)*
Previous read-only value.

Remarks

When read-only mode is set, characters typed at the keyboard do not get inserted into the MLE text. The API insertion interface, however, is still functional, as are selection-manipulation activities and copy-to-clipboard operations. This is useful as a means of preventing text modification (such as in a help system), and for providing a minimal blocking printing semaphore.

Default Processing

The default window procedure takes no action on this message, other than to set **reply** to NULL.

MLM_SETTEXTCOLOR

This message sets the text color.

Parameters

param1

color *(LONG)*
Color.

param2 *(BIT32)*
Reserved.

NULL Reserved value.

Returns

reply

oldcolor *(LONG)*
Color previously used.

Remarks

This message sets the color in which MLE text is to be drawn, and updates the display as necessary.

The color values are the same as those used by GpiSetColor.

Default Processing

The default window procedure takes no action on this message, other than to set **reply** to NULL.

MLM_SETTABSTOP

This message sets the pel interval at which tab stops are placed.

Parameters

param1

tab *(PIX)*
Pel interval for tab stops.

param2 *(BIT32)*
Reserved.

NULL Reserved value.

Returns

reply

tabset *(PIX)*
Success indicator:

<0	An error occurred.
Other	The value to which the width was set.

Remarks

This message fails if the reserved value is not zero.

This message can cause an overflow, see MLN_OVERFLOW.

Default Processing

The default window procedure takes no action on this message, other than to set **reply** to NULL.

MLM_SETTEXTLIMIT

This message sets the maximum number of bytes that a multi-line entry field control can contain.

Parameters

param1

size *(LONG)*
Maximum number of characters in MLE NO_TRANS format.

param2 *(BIT32)*
Reserved.

NULL Reserved value.

Returns

reply

flt *(ULONG)*
Success indicator:

0	Successful completion. Current text fits within the new limit.
Other	The number of bytes by which the current text exceeds the proposed limit. The limit is not changed.

Remarks

The multi-line entry field control window procedure responds to this message by limiting the text size to **size** bytes. Text size is calculated using the NO_TRANS format. Note that this is bytes and *not* characters; DBCS programmers should calculate accordingly.

This message returns 0 if the text limit exceeds or is equal to the existing text. Otherwise, it returns the number of bytes by which the text would have overflowed, and does not change the limit.

The default, which is unbounded, can be specified by entering a non-positive limit.

Default Processing

The default window procedure takes no action on this message, other than to set **reply** to NULL.

MLM_SETWRAP

This message sets the wrap flag.

Parameters

param1

wrap *(BOOL)*
New value for wrap flag.

param2 *(BIT32)*
Reserved.

NULL Reserved value.

Returns

reply

success *(BOOL)*
Success indicator:

TRUE Successful completion
FALSE An error occurred.

Remarks

The multi-line entry field control window procedure responds to this message by setting the word-wrap mode and updating the screen as appropriate.

When word-wrap is turned on, the text is wrapped to fit the formatting rectangle width. When word-wrap is turned off, the text is allowed to trail off to the right until it reaches an end-of-line marker.

Word-wrapping is defined as follows: 'words' are sequences of non-white-space characters (white-space characters are space, line break, and tab). When word-wrapping is enabled, the whole word must appear on one line within the formatting rectangle, unless the word by itself is too long to fit. In this case the word is split following the last character that fits, and the remainder starts a new line.

This definition then applies recursively to the remainder of the word. The word continues to be visible. For editing purposes (for example, for word-selection) the word is viewed as a single word drawn over multiple lines.

Blank characters are always accumulated onto the current line, even if they exceed the horizontal formatting dimension, that is, blanks are allowed to trail off the right-hand edge. Line-break characters are also allowed to exceed the horizontal dimension, and any subsequent text must begin on a new line. The line-break following a line-break character is sometimes referred to as a 'hard line-break'. Other line-breaks, due to word-wrapping, and not to explicit formatting characters, are referred to as 'soft line-breaks'.

Tab characters must always be visible. If a tab character occurs after the last tab stop within the horizontal formatting dimension, a soft line-break occurs after the tab.

This message can cause an overflow, see MLN_OVERFLOW.

Default Processing

The default window procedure takes no action on this message, other than to set **reply** to NULL.

MLM_UNDO

This message performs any available 'undo' operation.

Parameters

param1 *(BIT32)*
Reserved.

NULL Reserved value.

param2 *(BIT32)*
Reserved.

NULL Reserved value.

Returns

reply

undone *(USHORT)*
Success indicator:

TRUE	An 'undo' operation was performed
FALSE	No 'undo' operation was performed.

Remarks

The last operation is undone (note that an 'undo' can be undone).

This can cause an overflow, see MLN_OVERFLOW.

Default Processing

The default window procedure takes no action on this message, other than to set **reply** to NULL.

WM_BUTTON1DBLCLK

For the cause of this message, see "WM_BUTTON1DBLCLK" on page 12-6.

Parameters

For a description of the parameters, see "WM_BUTTON1DBLCLK" on page 12-6.

Remarks

This message indicates that mouse button 1 has clicked twice within the system double-click time.

Double-Click:

If the click point is in the middle of a non-white-space character, the token ('word') surrounding the clicked-on character, and any trailing spaces, are selected. If the click point is in a space character, the previous word (along with the trailing spaces including the clicked-on space) is selected. If there is no preceding word (either because the spaces are at the beginning of the text or immediately follow a line-break character) the run of spaces is selected. If the click point is on a tab or line-break character, that character is selected.

Shift-Double-Click:

Double-clicking while the Shift key is held down leaves the anchor point alone, and moves the cursor point to the beginning or end of the clicked-on token. If the click point is before the anchor point in the text, the cursor point is moved to the beginning of the surrounding word, otherwise, the cursor point is moved to the end of the surrounding word. When shift-double-clicking, the selection is extended to include the token that was double-clicked on.

Margin Mouse Event:

All mouse events in a margin cause the MLE to send a MLN_MARGIN notification to the MLE's owner window. This message has, as its parameters, the original mouse message. The owner can process the notification or not. If the owner does not process the message, the event is treated as if it occurred on the closest point in the text.

Default Processing

The default window procedure takes no action on this message, other than to set **reply** to NULL.

WM_BUTTON1DOWN

For the cause of this message, see "WM_BUTTON1DOWN" on page 12-7.

Parameters

For a description of the parameters, see "WM_BUTTON1DOWN" on page 12-7.

Remarks

This message delimits mouse button click events. Between a button-down and a button-up event the mouse is considered to be dragging. A mouse click is considered to happen on button-down, and dragging is terminated by a button-up.

Click:

Clicking in the text sets the cursor and anchor points to the nearest insertion point. If the MLE is in overtype mode, the anchor is extended one character further in the text, subject to the end-of-text and new-line boundary conditions, defined under WM_CHAR.

Shift-Click:

Clicking while the shift key is held down sets the cursor point to the nearest insertion point, while leaving the anchor point alone.

Margin Mouse Event:

All mouse events in a margin cause the MLE to send a MLN_MARGIN notification to the MLE's owner window. This message has, as its parameters, the original mouse message. The owner can process the notification or not. If the owner does not process the message, the event is treated as if it occurred on the closest point in the text.

Default Processing

The default window procedure takes no action on this message, other than to set **reply** to NULL.

WM_BUTTON1UP

For the cause of this message, see "WM_BUTTON1UP" on page 12-7.

Parameters

For a description of the parameters, see "WM_BUTTON1UP" on page 12-7.

Remarks

This message delimits mouse button click events. Between a button-down and a button-up event the mouse is considered to be dragging. A mouse click is considered to happen on button-down, and dragging is terminated by a button-up.

Margin Mouse Event:

All mouse events in a margin cause the MLE to send a MLN_MARGIN notification to the MLE's owner window. This message has, as its parameters, the original mouse message. The owner can process the notification or not. If the owner does not process the message, the event is treated as if it occurred on the closest point in the text.

Default Processing

The default window procedure takes no action on this message, other than to set **reply** to NULL.

WM_CHAR

For the cause of this message, see "WM_CHAR" on page 12-14.

Parameters

For a description of the parameters, see "WM_CHAR" on page 12-14.

Remarks

The behavior of the MLE, when typing, depends on whether it is in insert or overtype mode, and whether the selection is empty or not. The selection is defined to be 'empty' when the cursor point is equal to the anchor point.

When a character is typed, it replaces the current selection. If the selection is empty, the character is viewed as 'replacing nothing' so the character is effectively inserted into the text. If one or more characters are selected, those characters are deleted from the text and replaced by the typed character.

If the MLE is in insert mode, the cursor and anchor points are moved to immediately follow the newly-typed character.

If the MLE is in overtype mode, the cursor is moved to immediately follow the newly-typed character. If there is no character after the cursor (that is, the new character is at the end of the text) or if the character after the cursor is a line-break character, the anchor is set to be equal to the cursor point. In any other case, the anchor is extended one character past the cursor point, defining the next character as the current selection.

If the typing causes the cursor to go off the screen in any direction, the display is automatically scrolled. If word-wrap is on, text continues on a new line, otherwise, the screen is scrolled horizontally.

Scrolling of the text in the window is independent of cursor movement, but the cursor and selection remain unaltered at the same location within the text during all scrolling. The converse is not true. Any movement of the cursor will cause auto-scrolling, if necessary, to ensure that the text location of the cursor is visible within the window.

Tabs: Tabs are represented as a single character in the text model, and are displayed as enough white-space to reach to the next tab stop. Tabstops are set at pel intervals, starting with zero and occurring every n pels, where n is a value set by the MLM_SETTABSTOP message, and defaulting to eight times the average character width of the system font. When a tab is drawn, it uses the number of pels defined by the following formula:

```
pelWidth = pelTab - (pelDraw mod pelTab)
```

where pelTab is the tab interval, in pels, and pelDraw is the pel at which drawing is to begin.

Return: RETURN (ASCII newline) causes a hard line-break, and the following text begins on a new line. A line-break character is inserted in the text, which is drawn as a few pels of white-space (for selection purposes).

Key stroke commands: For all the following keys, unless otherwise noted, the display is scrolled, if necessary, to keep the cursor point visible. Where noted, the cursor setting behaves differently in insert mode than in overtype mode. This is subject to the conditions noted above.

DEL	Causes the contents of the selection region to be deleted. If the selection region contains no text, it causes the character to the right of the cursor to be deleted.
SHIFT+DEL	Causes the contents of the selection region to be cut to the clipboard.
INSERT	Toggles between insert and overtype mode. The MLE ignores the Insert key when it occurs without a modifier.
SHIFT+INS	Causes the contents of the clipboard to replace the selection region.
CTRL+INS	Causes the selection region to be copied to the clipboard. The selection region is not otherwise affected.

BACKSPACE Functions similar to DEL. If the selection is not empty, backspace deletes the selection. If the selection is empty, backspace deletes the character to the left of the cursor point. If the MLE is in overtype mode, the anchor point is set, and the cursor point is moved to be one character previous in the text. If no such character exists (because the anchor is set to the beginning of the text) the cursor is set to the anchor point. If the MLE is in insert mode, the cursor and anchor points are set as defined at the start of this chapter.

DOWN Sets the cursor point to the closest insertion point on the following line, then sets the anchor point to the cursor point (insertion mode) or one character following (overtype mode).

SHIFT+DOWN Causes the cursor point to be moved to the closest insertion point on the following line. The anchor point does not move.

UP Sets the cursor point to the closest insertion point on the preceding line, then sets the anchor point to the cursor point (insert mode) or one character following (overtype mode).

SHIFT+UP Sets the cursor point to the closest insertion point on the preceding line. The anchor point is not moved.

RIGHT Sets the cursor point to the insertion point one character following the cursor point. The anchor point is set to the cursor point (insert mode) or one character following (overtype mode).

SHIFT+RIGHT Causes the cursor point to be set to the insertion point immediately following the previous cursor point. The anchor point is not moved.

LEFT and SHIFT+LEFT Work analogously.

CTRL+RIGHT Moves the cursor point to the insertion point immediately preceding the next word in the text including trailing spaces, and sets the anchor point to be equal to (insert mode) or one character following (overtype mode) the cursor point. The EOL (hard line-break) and tab characters are treated as words.

CTRL+SHIFT+RIGHT Moves only the cursor point in the same way as CTRL+RIGHT but leaves the anchor point unmoved.

CTRL+LEFT Moves the cursor point to the preceding insertion point at the beginning of a word, and sets the anchor point to be equal to (insert mode) or one character following (overtype mode) the cursor point. The EOL (hard line-break) and tab characters are treated as words.

CTRL+SHIFT+LEFT Moves only the cursor point in the same way as CTRL + LEFT but leaves the anchor point unmoved.

PAGEDOWN and PAGEUP Cause the display to be scrolled by one screenful in either direction. This behavior is the same as would be encountered during a page-down or page-up caused by the scroll-bar.

CTRL+PAGEDOWN and CTRL+PAGEUP Cause the display to be scrolled by one screenful to the right or left respectively. This behavior is the same as would be encountered during a page-right or page-left caused by the scroll-bar.

HOME Sets the cursor point to the insertion point at the beginning of the line containing the cursor point, and sets the anchor point equal to (insert mode) or one character following (overtype mode).

SHIFT+HOME Moves the cursor point to the insertion point at the beginning of the line. The anchor point is not moved.

END Sets the anchor point to the insertion point at the end of the line containing the cursor point. If the last character on the line is a line-break character, the anchor is positioned just before it. The cursor is set equal to (insert mode) or one character previous to (overtype mode) the anchor.

SHIFT+END Moves the cursor point to the insertion point at the end of the line, as above. The anchor point is not moved.

CTRL+HOME Moves the cursor point to the insertion point at the beginning of the document. The anchor point is set equal to (insert mode) or one character following (overtype mode).

CTRL+END Moves the anchor point to the insertion point at the end of the document. The cursor point is set to be equal to the anchor point (insert mode) or one character preceding it (overtype mode).

CTRL+SHIFT+HOME Moves the cursor point in the same way as CTRL+HOME, but leaves the anchor point unmoved.

CTRL+SHIFT+END Moves the cursor point in the same way as CTRL+END, but leaves the anchor point unmoved.

Default Processing

The default window procedure takes no action on this message, other than to set **result** to FALSE.

WM_ENABLE

For the cause of this message, see "WM_ENABLE" on page 12-20.

Parameters

For a description of the parameters, see "WM_ENABLE" on page 12-20.

Remarks

The multi-line entry field control window procedure responds to this message by setting the enable state and by setting **reply** to NULL.

Disabling the window is similar, but not identical, to MLM_DISABLEREFRESH. Enabling the window is similar, but not identical, to MLM_ENABLEREFRESH. (Note that this also applies to window styles.) The difference is that a disabled window receives no mouse or keyboard input whereas with MLM_DISABLEREFRESH it receives the input but discards it.

Default Processing

The default window procedure takes no action on this message, other than to set **reply** to NULL.

WM_MOUSEMOVE

See "WM_MOUSEMOVE" on page 12-26 for the cause of this message.

Parameters

See "WM_MOUSEMOVE" on page 12-26 for a description of the parameters.

Remarks

The mouse pointer moves and is of interest to the MLE. If refresh is disabled, the pointer is set to the 'wait' icon (an hourglass). If refresh is enabled, the pointer is set to an 'I-beam'. This message can occur during dragging or when simply tracking the mouse.

Dragging:

Dragging sets the selection anchor to be the point where dragging begins, and moves the cursor point along with it as the mouse is moved. Moving the pointer into the margins while dragging effects a scroll in the appropriate direction and continues selecting.

Margin Mouse Event:

All mouse events in a margin cause the MLE to send a MLN_MARGIN notification to the MLE's owner window. This message has, as its parameters, the original mouse message. The owner can process the notification or not. If the owner does not process the message, the event is treated as if it occurred on the closest point in the text.

Default Processing

The default window procedure takes no action on this message, other than to set **reply** to NULL.

WM_QUERYWINDOWPARAMS

This message occurs when an application queries the entry field control window parameters.

Parameters

For a description of the parameters, see "WM_QUERYWINDOWPARAMS" on page 12-35.

Remarks

The multi-line entry field control window procedure responds to this message by returning the window parameters indicated by the **status** parameter of the *WNDPARAMS* data structure, identified by the **wndparams** parameter.

In response to the WPM_CCHTEXT flag the text length is reported in the CF_TEXT format. If it exceeds 64KB−1, this value is reported. In response to the WPM_TEXT flag, text up to the amount returned for the WPM_CCHTEXT value is placed at the indicated location in CF_TEXT format.

Default Processing

The default window procedure sets the **length**, **presparamslength**, and **ctldatalength** parameters of the *WNDPARAMS* data structure, identified by **wndparams**, to zero and sets **result** to FALSE.

WM_SETWINDOWPARAMS

This message occurs when an application sets or changes the entry field control window parameters.

Parameters

For a description of the parameters, see "WM_SETWINDOWPARAMS" on page 12-40.

Remarks

The multi-line entry field control window procedure responds to this message by setting the window parameters indicated by the **status** parameter of the *WNDPARAMS* data structure, identified by the **wndparams** parameter.

If the MLE text is to be set by this message, it is assumed to be in CF_TEXT format (see MLM_FORMAT) and all existing text is deleted before the new text is inserted. Note that a Control Data structure can be associated with the window parameters, in which case any field in that structure can cause a change to the MLE.

Default Processing

The default window procedure takes no action on this message, other than to set **result** to FALSE.

Chapter 19. Prompted Entry Field Control Window Processing

This system-provided window procedure processes the actions on a prompted entry field (combo box) control (WC_COMBOBOX).

Purpose

A combo box consists of an entry field control and a list box control merged into a single control. The list, which is usually limited in size, is displayed below the entry field, and offset one dialog-box unit to its right.

When the combo box control has the focus, the text in the entry field is given selected emphasis and, if the list box control has a matching entry, it is scrolled to show that match at the top of the list.

Combo Box Control Styles

These combo box control styles are available:

CBS_SIMPLE Both the entry field control and the list box control are visible. When the selection changes in the list box control, the text of the selected item in the list box control is placed in the entry field. Also, the text in the entry field is completed by extending the text of the entry field with the closest match from the list box.

CBS_DROPDOWN Inherits all the properties of a combo box control with a style of CBS_SIMPLE and, in addition, the list box control is hidden until the user requests that it should be displayed.

CBS_DROPDOWNLIST In which the entry field control is replaced by a static control that displays the current selection from the list box control. The user must explicitly cause the display of the list box control in order to make alternative selections in the list box.

Combo Box Control Data

None.

Combo Box Control Notification Messages

The combo box control uses the same messages as the entry field control and the list box control to notify its owner of significant events.

WM_CONTROL

For the cause of this message, see "WM_CONTROL" on page 12-17.

Parameters

param1

id *(USHORT)*
Control window identity.

notifycode *(USHORT)*
Notify code:

CBN_EFCHANGE	The content of the entry field control has changed, and the change has been displayed on the screen.
CBN_MEMERROR	The entry field control cannot allocate the storage necessary to accommodate window text of the length implied by the EM_SETTEXTLIMIT message.
CBN_EFSCROLL	The entry field control is about to scroll horizontally. This can happen in these circumstances: • The application has issued a WinScrollWindow call • The content of the entry field control has changed • The caret has moved • The entry field control must scroll to show the caret position.
CBN_LBSELECT	An item in the list box control has been selected.
CBN_LBSCROLL	The list box is about to scroll.
CBN_SHOWLIST	The list box is about to be displayed.
CBN_ENTER	The user has depressed the ENTER key or double clicked on an item in the list box control.

param2

controlspec *(HWND)*
Combo box control window handle.

Returns

reply *(BIT32)*
Reserved.

NULL Reserved value.

Remarks

The entry field control window procedure generates this message and sends it to its owner, informing the owner of the event.

Default Processing

The default window procedure takes no action on this message, other than to set **reply** to NULL.

Combo Box Control Window Messages

The combo box control uses the same messages as the entry field control and the list box control. In particular, the following messages are supported to achieve the functions of a combo box:

WM_SETWINDOWPARAMS	To set the text of the entry field.
WM_QUERYWINDOWPARAMS	To obtain the text of the entry field.
LM_QUERYITEMCOUNT	To obtain the count of items in the list box control.
LM_INSERTITEM	To insert an item into the list box control.
LM_SETTOPINDEX	To scroll the list box control so that the specified item is at the top.
LM_QUERYTOPINDEX	To obtain the index of the item at the top of the list box control.
LM_DELETEITEM	To delete an item from the list box control. If necessary, this also changes the content of the entry field to the item at the top of the list box control.
LM_SELECTITEM	To select a specified item in the list box control. Also, this changes the content of the entry field to the item at the top of the list box control and, if the list box control is not visible, causes the list box control to 'dropdown' below the entry field control.
LM_QUERYSELECTION	To obtain the current selection in the list box control.
LM_SETITEMTEXT	To change the text of an item in the list box control. If necessary, this also changes the content of the entry field control.
LM_QUERYITEMTEXT	To obtain the text of an item in the list box control.
LM_QUERYITEMTEXTLENGTH	To obtain the length of the text of an item in the list box control.
LM_SEARCHSTRING	To obtain the index of an item in the list box control containing a specified string.
LM_DELETEALL	To delete all the items in the list box control.
WM_VSCROLL	To scroll the list box control.
WM_ENABLE	To enable the combo box control to respond to input.
EM_QUERYFIRSTCHAR	To obtain the character displayed at the left edge of the entry field control.
EM_SETFIRSTCHAR	To scroll the entry field control so that the specified character is displayed at the left edge of the entry field control.
EM_QUERYCHANGED	To obtain the changes to the entry field control.
EM_QUERYSEL	To obtain the current selection of the entry field control.
EM_SETSEL	To set the current selection of the entry field control.
EM_SETTEXTLIMIT	To set the maximum number of characters to be contained in the entry field control.
LM_SETSELECTION	To set the selection of the list box control without affecting the association of the cursor with an item in the list box control.
LM_QUERYCURSOR	To obtain the item currently associated with the cursor in the list box control.
LM_SETCURSOR	To set the item in the list box control to be associated with the cursor.
EM_CUT	To place the contents of the selection of the entry field control into the clipboard and then delete those contents from the entry field control.

EM_PASTE	To place the contents of the clipboard into the entry field control.
EM_COPY	To place the contents of the selection of the entry field control into the clipboard.
EM_CLEAR	To clear the current selection of the entry field control.

This section describes the combo box control window procedure actions on receiving these messages:

CBM_HILITE

This message sets the highlighting state of the entry field control.

Parameters

param1

hilite *(BOOL)*
Highlighting indicator:

TRUE	Highlight the entry field control
FALSE	Do not highlight the entry field control.

param2 *(BIT32)*
Reserved.

NULL	Reserved value.

Returns

reply

changed *(BOOL)*
Changed indicator:

TRUE	The highlighting state of the entry field has been changed
FALSE	The highlighting state of the entry field has not been changed.

Remarks

The combo box control window procedure responds to this message by setting the highlighting state of the entry field control.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set **changed** to the default value of NULL, which is equivalent to FALSE.

CBM_ISLISTSHOWING

This message determines if the list box control is showing.

Parameters

param1 *(BIT32)*
Reserved.

NULL	Reserved value.

param2 *(BIT32)*
Reserved.

NULL	Reserved value.

Returns

reply

showing *(BOOL)*
Showing indicator:

TRUE The list box control is showing
FALSE The list box control is not showing.

Remarks

The combo box control window procedure responds to this message by indicating if the list box control is showing.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set **showing** to the default value of NULL, which is equivalent to FALSE.

CBM_SHOWLIST

This message sets the showing state of the list box control.

Parameters

param1

showing *(BOOL)*
Showing indicator:

TRUE Show the list box control
FALSE Do not show the list box control.

param2 *(BIT32)*
Reserved.

NULL Reserved value.

Returns

reply

changed *(BOOL)*
Changed indicator:

TRUE The list box showing state has been changed
FALSE The list box showing state has not been changed.

Remarks

The combo box control window procedure responds to this message by setting the showing state of the list box control.

This message has no affect on a combo box control whose style is CBS_SIMPLE.

Hiding the list box control has no affect on the selection in the list box control. The selection in the list box control must be changed by the use of a LM_SELECTITEM message.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set **changed** to the default value of NULL, which is equivalent to FALSE.

WM_UPDATESTYLE

This message sets the style bits specified by the **mask** parameter to the values specified in the **data** parameter.

Parameters

param1

data *(BIT32)*
Style data.

param2

mask *(BIT32)*
Style mask.

Returns

reply

success *(BOOL)*
Success indicator:

TRUE	Successful completion
FALSE	Error occurred.

Remarks

The combo box control window procedure responds to this message by setting the style bits and then redrawing the window to reflect the updated style.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set **success** to the default value of NULL, which is equivalent to FALSE.

Chapter 20. Scroll Bar Control Window Processing

This system-provided window procedure processes the actions on a scroll bar control (WC_SCROLLBAR).

Scroll Bar Control Styles

These scroll bar control styles are available:

SBS_HORZ Create a horizontal scroll bar.
SBS_VERT Create a vertical scroll bar.
SBS_SETTHUMBSIZE Indicates the presence of the **visible** and **total** parameters in the *SBCDATA* data structure.

Scroll Bar Control Data

SBCDATA Scroll-bar control data structure.

length *(COUNT2B)*
Length of control data in bytes.

10 The length of the control data for a scroll-bar control.

hilite *(USHORT)*
Highlighting code.

This indicates which part of the scroll bar is to be highlighted, if any.

ZERO No highlighting.
SB_LINEUP Line up arrow.
SB_LINELEFT Line left arrow.
SB_LINEDOWN Line down arrow.
SB_LINERIGHT Line right arrow.
SB_PAGEUP Page up arrow.
SB_PAGELEFT Page left arrow.
SB_PAGEDOWN Page down arrow.
SB_PAGERIGHT Page right arrow.
SB_SLIDERTRACK Slider.

first *(SHORT)*
First bound of the scroll-bar range.

last *(SHORT)*
Last bound of the scroll-bar range.

slider *(SHORT)*
Slider position.

visible *(SHORT)*
Number of data items visible.

total *(SHORT)*
Number of data items available.

Scroll Bar Control Notification Messages

These messages are initiated by the scroll bar control window procedure to notify its owner of significant events.

WM_HSCROLL

This message occurs when a horizontal scroll bar control has a significant event to notify to its owner.

Parameters

param1

identifier *(USHORT)*
Scroll bar control window identifier.

param2

slider *(SHORT)*
Slider position:

0	Either the operator is not moving the slider with the pointer device, or for the instance where **cmd** is SB_SLIDERPOSITION the pointer is outside the tracking rectangle when the button is released.
Other	Slider position.

cmd *(USHORT)*
Command:

SB_LINELEFT	Sent if the operator clicks on the left arrow of the scroll bar, or depresses the VK_LEFT key.
SB_LINERIGHT	Sent if the operator clicks on the right arrow of the scroll bar, or depresses the VK_RIGHT key.
SB_PAGELEFT	Sent if the operator clicks on the area to the left of the slider, or depresses the VK_PAGELEFT key.
SB_PAGERIGHT	Sent if the operator clicks on the area to the right of the slider, or depresses the VK_PAGERIGHT key.
SB_SLIDERPOSITION	Sent to indicate the final position of the slider.
SB_SLIDERTRACK	If the operator moves the scroll bar slider with the pointer device, this is sent every time the slider position changes.
SB_ENDSCROLL	Sent when the operator has finished scrolling, but only if the operator has not been doing any absolute slider positioning.

Returns

reply *(BIT32)*
Reserved.

NULL	Reserved value.

Remarks

The scroll bar control window procedure generates this message and posts it to its owner, informing the owner of the event.

Default Processing

The default window procedure takes no action on this message, other than to set **reply** to NULL.

WM_VSCROLL

This message occurs when a vertical scroll bar control has a significant event to notify to its owner.

Parameters

param1

identifier *(USHORT)*
Scroll bar control window identifier.

param2

slider *(SHORT)*
Slider position:

0	Either the operator is not moving the slider with the pointer device, or for the instance when **cmd** is SB_SLIDERPOSITION the pointer is outside the tracking rectangle when the button is released.
Other	Slider position.

cmd *(USHORT)*
Command:

SB_LINEUP	Sent if the operator clicks on the up arrow of the scroll bar, or presses the VK_UP key.
SB_LINEDOWN	Sent if the operator clicks on the down arrow of the scroll bar, or presses the VK_DOWN key.
SB_PAGEUP	Sent if the operator clicks on the area above the slider, or presses the VK_PAGEUP key.
SB_PAGEDOWN	Sent if the operator clicks on the area below the slider, or presses the VK_PAGEDOWN key.
SB_SLIDERPOSITION	Sent to indicate the final position of the slider.
SB_SLIDERTRACK	If the operator moves the scroll bar slider with the pointer device, this is sent every time the slider position changes.
SB_ENDSCROLL	Sent when the operator has finished scrolling, but only if the operator has not been doing any absolute slider positioning.

Returns

reply *(BIT32)*
Reserved.

NULL	Reserved value.

Remarks

The scroll bar control window procedure generates this message and posts it to its owner, informing the owner of the event.

Default Processing

The default window procedure takes no action on this message, other than to set **reply** to NULL.

Scroll Bar Control Window Messages

This section describes the scroll bar control window procedure actions on receiving the following messages.

SBM_QUERYHILITE

This message queries a scroll bar's highlighting state.

Parameters

param1 *(BIT32)*
Reserved.

NULL Reserved value.

param2 *(BIT32)*
Reserved.

NULL Reserved value.

Returns

reply

code *(SHORT)*
Highlighting state:

SB_LINEUP	Line up/line left arrow.
SB_LINEDOWN	Line down/line right arrow.
SB_PAGEUP	Page up/page left area.
SB_PAGEDOWN	Page down/page right area.
SB_SLIDERTRACK	Thumb.

Remarks

The scroll bar control responds to this message by returning the highlighting state.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, except to set **code** to the default value of NULL which is equivalent to zero.

SBM_QUERYPOS

This message returns the slider position.

Parameters

param1 *(BIT32)*
Reserved.

NULL Reserved value.

param2 *(BIT32)*
Reserved.

NULL Reserved value.

Returns

reply

slider *(SHORT)*
Slider position.

Remarks

The scroll bar control window procedure responds to this message by returning the slider position.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set **slider** to the default value of NULL, which is equivalent to zero.

SBM_QUERYRANGE

This message returns the scroll bar range.

Parameters

param1 *(BIT32)*
Reserved.

NULL Reserved value.

param2 *(BIT32)*
Reserved.

NULL Reserved value.

Returns

reply

first *(SHORT)*
First bound.

last *(SHORT)*
Last bound.

Remarks

The scroll bar control window procedure responds to this message by returning the first and last bounds of the scroll bar range.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set **reply** to the default value of NULL, which is equivalent to setting both **first** and **last** to zero.

SBM_SETHILITE

This message sets a scroll bar's highlighting state.

Parameters

param1

code *(SHORT)*
New highlighting state:

SB_LINEUP	Line up/line left arrow.
SB_LINEDOWN	Line down/line right arrow.
SB_PAGEUP	Page up/page left area.
SB_PAGEDOWN	Page down/page right area.
SB_SLIDERTRACK	Thumb.

reserved *(BIT16)*
Reserved.

NULL Reserved value.

param2 *(BIT32)*
Reserved.

NULL Reserved value.

Returns

reply

success *(BOOL)*
Success indicator:

TRUE The highlighting state is set successfully
FALSE The highlighting state is not set.

reserved *(BIT16)*
Reserved.

NULL Reserved value.

Remarks

The scroll bar control responds to this message by setting the highlighting state.

The scroll bar control is redrawn to reflect the change.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, except to set **success** to the default value of NULL which is equivalent to FALSE.

SBM_SETPOS

This message sets the position of the slider.

Parameters

param1

slider *(SHORT)*
Position of slider.

If this value is outside the scroll-bar range, the slider is moved to the nearest valid position within the range.

param2 *(BIT32)*
Reserved.

NULL Reserved value.

Returns

reply

success *(BOOL)*
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Remarks

The scroll bar control window procedure responds to this message by setting the position of the slider.

The scroll bar control is redrawn to reflect the change.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set **success** to the default value of NULL, which is equivalent to FALSE.

SBM_SETSCROLLBAR

This message sets the scroll bar range and slider position.

Parameters

param1

slider *(SHORT)*
Position of slider.

If this value is outside the scroll-bar range, the slider is moved to the nearest valid position within the range.

param2

first *(SHORT)*
First bound.

This value must not be less than zero. If a value less than zero is supplied, zero is used as the value.

last *(SHORT)*
Last bound.

The value must not be less than zero or **first**. If a value less than this is supplied, the higher of zero or **first** is used as the value.

Returns

reply

success *(BOOL)*
Success indicator:

TRUE	Successful completion
FALSE	Error occurred.

Remarks

The scroll bar control window procedure responds to this message by setting the values of the information range and the position of the slider.

The scroll bar is redrawn to reflect the change.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set **success** to the default value of NULL, which is equivalent to FALSE.

SBM_SETTHUMBSIZE

This message sets the scroll bar slider size.

Parameters

param1

visible *(SHORT)*
Size of the visible part of the document.

total *(SHORT)*
Size of the entire document.

param2 *(BIT32)*
Reserved.

NULL	Reserved value.

Returns

reply

success *(BOOL)*
Success indicator:

TRUE	Successful completion
FALSE	Error occurred.

Remarks

The scroll bar control window procedure responds to this message by setting the size of the slider proportional to the visible part of the document. If the visible part exceeds or is equal to the entire document the scroll bar is disabled, otherwise the scroll bar is enabled.

The scroll bar is redrawn to reflect the change.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set **success** to the default value of NULL, which is equivalent to FALSE.

WM_QUERYCONVERTPOS

For the cause of this message, see "WM_QUERYCONVERTPOS" on page 12-33.

Parameters

For a description of the parameters, see "WM_QUERYCONVERTPOS" on page 12-33.

Remarks

The scroll bar control window procedure returns QCP_NOCONVERT.

Default Processing

For the default window procedure processing of this message see "WM_QUERYCONVERTPOS" on page 12-33.

WM_QUERYWINDOWPARAMS

This message occurs when an application queries the scroll bar control window parameters.

Parameters

For a description of the parameters, see "WM_QUERYWINDOWPARAMS" on page 12-35.

Remarks

The scroll bar control window procedure responds to this message by returning the window parameters indicated by the **status** parameter of the *WNDPARAMS* data structure identified by the **wndparams** parameter.

Default Processing

The default window procedure sets the **length**, **presparamslength**, and **ctldatalength** parameters of the *WNDPARAMS* data structure, identified by **wndparams**, to zero and sets **result** to FALSE.

WM_SETWINDOWPARAMS

This message occurs when an application sets or changes the scroll bar control window parameters.

Parameters

For a description of the parameters, see "WM_SETWINDOWPARAMS" on page 12-40.

Remarks

The scroll bar control window procedure responds to this message by setting the window parameters indicated by the **status** parameter of the *WNDPARAMS* data structure identified by the **wndparams** parameter.

Default Processing

The default window procedure takes no action on this message, other than to set **result** to FALSE.

Chapter 21. Static Control Window Processing

This system-provided window procedure processes the actions on a static control (WC_STATIC).

Static Control Styles

Static controls can have these styles:

SS_TEXT Creates a text field. The text is formatted before it is displayed according to the setting of these text drawing-style flags:

Flag	Meaning
DT_LEFT	Left-justified text
DT_CENTER	Centered text
DT_RIGHT	Right-justified text

ORed with one of:

Flag	Meaning
DT_TOP	Text is aligned to top of window
DT_VCENTER	Text is aligned vertically in center of window
DT_BOTTOM	Text is aligned to bottom of window.

The following text drawing style can also be ORed, but only if DT_TOP and DT_LEFT are also specified:

DT_WORDBREAK Text is multi-line with word-wrapping at ends of lines.

Programming Note: For "static" text that can be selected, a Button Control with a style of BS_NOBORDER can be used.

SS_GROUPBOX A group box static control is a box that has an identifying text string in its upper left corner. Group boxes are used to collect a group of radio buttons or other controls into a single unit.

SS_ICON Draws an icon. The text of the static control is a string that is used to derive the resource ID from which the icon is loaded. The format of the string is:

- The first byte is X'FF', the second byte is the low byte of the resource ID, and the third byte is the high byte of the resource ID.
- The first character is "#"; subsequent characters make up the decimal text representation of the resource ID.

 This format can be used for specifying a system icon in a resource file. The decimal string is the value of the appropriate SPTR_* constant.

If the string is empty or does not follow the format above, no resource is loaded.

The resource is assumed to reside in the resource file of the current process.

This control is resized to the size of the icon.

SS_SYSICON This style is the same as SS_ICON except that the icon ID is specified as one of the system pointer ID values (SPTR_* values) rather than a resource ID. This style provides a convenient way to include system icons in application dialog boxes.

SS_BITMAP Draws a bit map. The text of the static control names the bit-map resource, as for SS_ICON.

SS_FGNDRECT Creates a foreground color-filled rectangle.

SS_BKGNDRECT Creates a background color-filled rectangle.

SS_FGNDFRAME Creates a box with a foreground color frame.

SS_BKGNDFRAME Creates a box with a background color frame.

Static Control Data

None.

Static Control Notification Messages

No notification messages are initiated by the static control window procedure.

Static Control Window Messages

This section describes the static control window procedure actions on receiving the following messages.

SM_QUERYHANDLE

This message returns the icon or bit-map handle of a static control.

Parameters

param1 *(BIT32)*
Reserved.

NULL Reserved value.

param2 *(BIT32)*
Reserved.

NULL Reserved value.

Returns

reply

handle *(HBITMAP)*
Icon or bit-map handle of the static control:

NULL No icon or bit-map handle of the static control exists, or an error occurred.
Other Icon or bit-map handle of the static control.

Remarks

The static control window procedure responds to this message by setting **handle** to the handle of the icon or bit map of the static control.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set **handle** to the default value of NULL.

SM_SETHANDLE

This message sets the icon or bit-map handle of a static control.

Parameters

param1

handle *(HBITMAP)*
Icon or bit-map handle of a static control.

This is an icon handle when sent to a control with a style of SS_ICON or SS_SYSICON, and a bit-map handle when sent to a control with a style of SS_BITMAP.

param2 *(BIT32)*
Reserved.

NULL Reserved value.

Returns

reply

success *(BOOL)*
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Remarks

The static control window procedure responds to this message by setting the icon or bit-map handle of a static control to the value specified by **handle**, and causes the static control to be redrawn, using the new item handle.

It should only be sent to a control with a style of SS_BITMAP, SS_ICON, or SS_SYSICON.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set **success** to the default value of NULL, which is equivalent to FALSE.

WM_MATCHMNEMONIC

For the cause of this message, see "WM_MATCHMNEMONIC" on page 12-52.

Parameters

For a description of the parameters, see "WM_MATCHMNEMONIC" on page 12-52.

Remarks

The static control window procedure responds to this message by setting **result** as appropriate.

Default Processing

The default window procedure takes no action on this message, other than to set **result** to FALSE.

WM_QUERYCONVERTPOS

For the cause of this message, see "WM_QUERYCONVERTPOS" on page 12-33.

Parameters

For a description of the parameters, see "WM_QUERYCONVERTPOS" on page 12-33.

Remarks

The static control window procedure returns QCP_NOCONVERT.

Default Processing

For the default window procedure processing of this message see "WM_QUERYCONVERTPOS" on page 12-33.

WM_QUERYWINDOWPARAMS

This message occurs when an application queries the static control window procedure window parameters.

Parameters

For a description of the parameters, see "WM_QUERYWINDOWPARAMS" on page 12-35.

Remarks

The static control window procedure responds to this message by passing it to the default window procedure.

Default Processing

The default window procedure sets the **length**, **presparamslength**, and **ctldatalength** parameters of the *WNDPARAMS* data structure, identified by **wndparams**, to zero and sets **result** to FALSE.

WM_SETWINDOWPARAMS

This message occurs when an application sets or changes the static control window procedure window parameters.

Parameters

For a description of the parameters, see "WM_SETWINDOWPARAMS" on page 12-40.

Remarks

The static control window procedure responds to this message by passing it to the default window procedure.

Default Processing

The default window procedure takes no action on this message, other than to set **result** to FALSE.

Chapter 22. Title Bar Control Window Processing

This system-provided window procedure processes the actions on a title-bar control (WC_TITLEBAR).

Purpose

The title bar control is the frame control that is used to display the application window title. It is also used to display the active or inactive status of the frame window.

The title-bar control also implements the user interface for moving the frame window.

The standard identifier for a title bar control in a frame window is FID_TITLEBAR.

Title Bar Control Styles

There is only one title-bar style, the default.

Title Bar Control Data

None.

Title Bar Control Notification Messages

These messages are initiated by the title-bar control to notify its owner of significant events.

WM_COMMAND

For the cause of this message, see "WM_COMMAND" on page 12-16.

Parameters

For a description of the parameters, see "WM_COMMAND" on page 12-16.

The title bar control window procedure sets **cmd** to the title bar control identity and **source** to CMDSRC_OTHER.

Remarks

The title bar control window procedure generates this message and posts it to the queue of its owner, when a WM_BUTTON1DBLCLK is received.

Default Processing

The default window procedure takes no action on this message, other than to set **reply** to NULL.

WM_QUERYTRACKINFO

For the cause of this message, see "WM_QUERYTRACKINFO" on page 12-34.

Parameters

For a description of the parameters, see "WM_QUERYTRACKINFO" on page 12-34.

Remarks

The title bar control window procedure generates this message and sends it to its owner, informing the owner of this event, and acts on **result** as appropriate.

Default Processing

The default window procedure takes no action on this message, other than to set **result** to FALSE.

Title Bar Control Window Messages

This section describes the title bar control window procedure actions on receiving the following messages.

TBM_QUERYHILITE

This message returns the highlighting state of a title-bar control.

Parameters

param1 *(BIT32)*
Reserved.

NULL Reserved value.

param2 *(BIT32)*
Reserved.

NULL Reserved value.

Returns

reply

highlighted *(BOOL)*
Highlighting state:

TRUE Title-bar control is highlighted
FALSE Title-bar control is not highlighted.

Remarks

The title bar control window procedure responds to this message by returning the highlighting state of the title-bar window.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set **highlighted** to the default value of NULL, which is equivalent to FALSE.

TBM_SETHILITE

This message is used to highlight or unhighlight a title-bar control.

Parameters

param1

highlighted *(BOOL)*
Highlighting indicator:

TRUE Highlight the title-bar control
FALSE Remove highlight from the title-bar control.

param2 *(BIT32)*
Reserved.

NULL Reserved value.

Returns

reply

success *(BOOL)*
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Remarks

The title bar control window procedure responds to this message by setting the highlighting state according to **highlighted**.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set **success** to the default value of NULL, which is equivalent to FALSE.

TBM_TRACKMOVE

This message moves the owner window of the title bar.

Parameters

param1

tflags *(USHORT)*
Tracking flags.

Contains a combination of one or more TF_flags; for details, see the *TRACKINFO* data structure.

param2 *(BIT32)*
Reserved.

NULL Reserved value.

Returns

reply

success *(BOOL)*
Success indicator:

TRUE Successful completion
FALSE Error occurred, or the operation is terminated.

Remarks

The title bar control window procedure responds to this message from an application by moving its owner window.

A WM_QUERYTRACKINFO message is first sent to the owner of the size control. If the return value is TRUE, the sizing operation is performed using the *TRACKINFO* structure; otherwise the operation is terminated.

Default Processing

The default window procedure does not expect to receive this message and therefore takes no action on it, other than to set **success** to the default value of NULL, which is equivalent to FALSE.

WM_QUERYCONVERTPOS

For the cause of this message, see "WM_QUERYCONVERTPOS" on page 12-33.

Parameters

For a description of the parameters, see "WM_QUERYCONVERTPOS" on page 12-33.

Remarks

The title bar control window procedure returns QCP_NOCONVERT.

Default Processing

For the default window procedure processing of this message see "WM_QUERYCONVERTPOS" on page 12-33.

WM_QUERYWINDOWPARAMS

This message occurs when an application queries the title bar control window procedure window parameters.

Parameters

For a description of the parameters, see "WM_QUERYWINDOWPARAMS" on page 12-35.

Default Processing

The title bar control window procedure queries the appropriate window parameters in accordance with **wndparams** and sets **result** to TRUE if the operation is successful, otherwise to FALSE.

WM_SETWINDOWPARAMS

This message occurs when an application sets or changes the title bar control window procedure window parameters.

Parameters

For a description of the parameters, see "WM_SETWINDOWPARAMS" on page 12-40.

Default Processing

The title bar control window procedure sets the appropriate window parameters in accordance with **wndparams** and sets **result** to TRUE if the operation is successful, otherwise to FALSE.

Chapter 23. Clipboard Messages

WM_DESTROYCLIPBOARD

This message is sent to the clipboard owner when the clipboard is emptied through a call to WinEmptyClipbrd.

Parameters

param1 *(BIT32)*
Reserved.

Zero Reserved value.

param2 *(BIT32)*
Reserved.

NULL Reserved value.

Returns

reply *(BIT32)*
Reserved.

Zero Reserved value.

Remarks

If there is any data that has been set with the CFI_OWNERFREE flag, the clipboard owner must release the data at this time.

Default Processing

None.

WM_DRAWCLIPBOARD

This message is sent to the clipboard viewer window whenever the contents of the clipboard change; that is, as a result of the WinCloseClipbrd call following a call to WinSetClipbrdData.

Parameters

param1 *(BIT32)*
Reserved.

NULL Reserved value.

param2 *(BIT32)*
Reserved.

NULL Reserved value.

Returns

reply *(BIT32)*
Reserved.

NULL Reserved value.

Default Processing

None.

WM_HSCROLLCLIPBOARD

This message is sent to the clipboard-owner window when the clipboard contains a data handle for the CFI_OWNERDISPLAY format.

Parameters

param1

hwndviewer *(HWND)*
Handle.

This contains a handle to the clipboard application window.

param2

posscroll *(SHORT)*
Scroll position.

The position is either:

Zero **codescroll** is other than SB_SLIDERPOSITION
Other The position of the slider when **codescroll** is SB_SLIDERPOSITION.

codescroll *(SHORT)*
Scroll-bar code

This is one of the SB_* scroll-bar codes as defined in "WM_HSCROLL" on page 20-2.

SB_LINELEFT	Sent if the operator clicks the left arrow of the scroll bar, or depresses the VK_LEFT key.
SB_LINERIGHT	Sent if the operator clicks the right arrow of the scroll bar, or depresses the VK_RIGHT key.
SB_PAGELEFT	Sent if the operator clicks the area to the left of the slider, or depresses the VK_PAGELEFT key.
SB_PAGERIGHT	Sent if the operator clicks the area to the right of the slider, or depresses the VK_PAGERIGHT key.
SB_SLIDERPOSITION	Sent to indicate the final position of the slider.
SB_SLIDERTRACK	If the operator moves the scroll bar slider with the pointer device, this is sent every time the slider position changes.
SB_ENDSCROLL	Sent when the operator has finished scrolling, but only if the operator has not been doing any absolute slider positioning.

Returns

reply *(BIT32)*
Reserved.

NULL Reserved value.

Remarks

The clipboard owner is responsible for displaying the clipboard contents. The clipboard owner should use WinInvalidateRect or repaint as desired. The scroll-bar position is also reset.

Default Processing

None.

WM_PAINTCLIPBOARD

This message is sent when the clipboard contains a data handle with the CFI_OWNERDISPLAY information flag set.

Parameters

param1

hwndviewer *(HWND)*
Handle.

This is a handle to the clipboard application window.

param2 *(BIT32)*
Reserved.

NULL Reserved value.

Returns

reply *(BIT32)*
Reserved.

NULL Reserved value.

Remarks

As the clipboard owner is responsible for displaying the clipboard contents, this message notifies the clipboard application that its client area needs repainting. The WM_PAINTCLIPBOARD message is sent to the owner of the clipboard to request repainting of all or part of the clipboard application's client area.

Programming Note: To determine whether the entire client area needs repainting or just a portion of it, the clipboard owner must compare the dimensions of the drawing area to the dimensions given in the most recent WM_SIZECLIPBOARD message.

Default Processing

None.

WM_RENDERALLFMTS

This message is sent to the application that owns the clipboard whilst the application is being destroyed.

Parameters

param1 *(BIT32)*
Reserved.

Zero Reserved value.

param2 *(BIT32)*
Reserved.

NULL Reserved value.

Returns

reply *(BIT32)*
Reserved.

Zero Reserved value.

Remarks

The application renders the clipboard data in all formats it is capable of generating and passes a handle to each format to WinSetClipbrdData. This ensures that the data in the clipboard can be rendered even though the application has been destroyed.

Default Processing

None.

WM_RENDERFMT

This message is a request to the clipboard owner to render the data of the format specified in **fmt**.

Parameters

param1

fmt *(USHORT)*
Data format.

This is the format of the data to be rendered.

param2 *(BIT32)*
Reserved.

NULL Reserved value.

Returns

reply *(BIT32)*
Reserved.

NULL Reserved value.

Remarks

The data is rendered into a global handle, which is then set into the clipboard with WinSetClipbrdData.

Default Processing

None.

WM_SIZECLIPBOARD

This message is sent when the clipboard contains a data handle for the CFI_OWNERDISPLAY format, and the clipboard application window has changed size.

Parameters

param1

viewer *(HWND)*
Handle of viewer window.

param2

paint *(PRECTL)*
Rectangle to be painted.

Returns

reply *(BIT32)*
Reserved.

NULL Reserved value, must be null.

Default Processing

The default window procedure takes no action on this message except to set **reply** to NULL.

WM_VSCROLLCLIPBOARD

This message is sent to the clipboard owner window when the clipboard contains a data handle for the CFI_OWNERDISPLAY format.

Parameters

param1

hwndviewer *(HWND)*
Handle.

This contains a handle to the clipboard application window.

param2

posscroll *(SHORT)*
Scroll position.

The position is either:

Zero **codescroll** is other than SB_SLIDERPOSITION
Other The position of the slider when **codescroll** is SB_SLIDERPOSITION.

codescroll *(SHORT)*
Scroll-bar code.

This is one of the SB_* scroll-bar codes as defined in "WM_HSCROLL" on page 20-2.

SB_LINELEFT	Sent if the operator clicks the left arrow of the scroll bar, or depresses the VK_LEFT key.
SB_LINERIGHT	Sent if the operator clicks the right arrow of the scroll bar, or depresses the VK_RIGHT key.
SB_PAGELEFT	Sent if the operator clicks the area to the left of the slider, or depresses the VK_PAGELEFT key.
SB_PAGERIGHT	Sent if the operator clicks the area to the right of the slider, or depresses the VK_PAGERIGHT key.
SB_SLIDERPOSITION	Sent to indicate the final position of the slider.
SB_SLIDERTRACK	If the operator moves the scroll bar slider with the pointer device, this is sent every time the slider position changes.
SB_ENDSCROLL	Sent when the operator has finished scrolling, but only if the operator has not been doing any absolute slider positioning.

Returns

reply *(BIT32)*
Reserved.

NULL Reserved value.

Remarks

The clipboard owner is responsible for displaying the clipboard contents. The clipboard owner should use WinInvalidateRect or repaint as desired. The scroll bar position is also reset.

Default Processing

None.

Chapter 24. Dynamic Data Exchange Messages

This section describes the message part of the DDE protocol, which is a set of guidelines that allows two applications to share data freely between one another; not necessarily driven directly by user input.

Note: DDE operates between two specific applications, each of which must be aware of the other, and active.

WinDdeInitiate, WinDdePostMsg, and WinDdeRespond are the function calls associated with these messages.

WM_DDE_ACK

This message notifies an application of the receipt and processing of a WM_DDE_EXECUTE, WM_DDE_DATA, WM_DDE_ADVISE, WM_DDE_UNADVISE or WM_DDE_POKE message, and in some cases, of a WM_DDE_REQUEST message.

This message is always posted.

Parameters

param1

hwnd *(HWND)*
Sender's window handle.

param2

ddestruct *(PDDESTRUCT)*
DDE structure.

This points to a dynamic data exchange structure. See DDESTRUCT on page 2-6.

The acknowledging application modifies the **status** field to return information about the status of the message received:

DDE_FACK	1 = request accepted, 0 = request not accepted
DDE_FBUSY	1 = busy, 0 = not busy
DDE_NOTPROCESSED	Reserved for application-specific return codes
DDE_FAPPSTATUS	The message was not understood and was ignored.

An application is expected to set DDE_FBUSY if it is unable to respond to the request at the time it is received. The DDE_FBUSY flag is defined only when DDE_FACK is 0.

itemname identifies the item for which the acknowledgment is being sent.

Returns

reply *(BIT32)*
Reserved.

NULL Reserved Value.

Default Processing

None.

WM_DDE_ADVISE

This message (posted by a client application) requests the receiving application to supply an update for a data item whenever it changes.

This message is always posted.

Parameters

param1

hwnd *(HWND)*
Sender's window handle.

param2

ddestruct *(PDDESTRUCT)*
DDE structure.

This points to a dynamic data exchange structure. See DDESTRUCT on page 2-6.

Flags in the **status** field are set as follows:

DDE_FACKREQ	If this bit is 1, the receiving (server) application is requested to send its WM_DDE_DATA messages with the acknowledgment-requested (DDE_FACKREQ) bit set. This offers a flow control technique, whereby the client application can avoid overload from incoming WM_DDE_DATA messages.
DDE_FNODATA	If this bit is 1, the server is requested to send its WM_DDE_DATA messages with a zero length data portion. These messages are basically "alarms" telling the client that the source data has changed. Upon receiving one of these alarms, the client can choose to call for the latest version of the data by issuing a WM_DDE_REQUEST message, or it can choose to ignore the alarm altogether. This is typically used when there is a significant resource cost associated with actually rendering and/or assimilating the data.

itemname identifies which data item is being requested.

format is the client's preferred type of data. It must be a registered DDE data format number.

Returns

reply *(BIT32)*
Reserved.

NULL Reserved Value.

Remarks

The receiving application is expected to reply with a positive WM_DDE_ACK message if it can provide the requested data, or with a negative one if it can not.

Default Processing

None.

WM_DDE_DATA

This message notifies a client application of the availability of data. It is always posted.

Parameters

param1

hwnd *(HWND)*
Sender's window handle.

param2

ddestruct *(PDDESTRUCT)*
DDE structure.

This points to a dynamic data exchange structure. See DDESTRUCT on page 2-6.

Flags in the **status** field are set as follows:

DDE_FACKREQ	If this bit is 1, the receiving (client) application is expected to send a WM_DDE_ACK message after the memory object has been processed. If it is 0, the client application should not send a WM_DDE_ACK message.
DDE_FRESPONSE	If this bit is 1, this data is offered in response to a WM_DDE_REQUEST message. If it is 0, this data is offered in response to a WM_DDE_ADVISE message.

itemname identifies which data item is available.

data is the data. The format of the data is a registered DDE data format, identified by the **format** field.

Returns

reply *(BIT32)*
Reserved.

NULL Reserved value.

Default Processing

None.

WM_DDE_EXECUTE

This message posts a string to a server application to be processed as a series of commands. The server application is expected to post a WM_DDE_ACK message in response.

This message is always posted.

Parameters

param1

hwnd *(HWND)*
Sender's window handle.

param2

ddestruct *(PDDESTRUCT)*
DDE structure.

This points to a dynamic data exchange structure. See DDESTRUCT on page 2-6.

data contains the command(s) to be executed.

Returns

reply *(BIT32)*

Reserved.

NULL Reserved Value.

Default Processing

None.

WM_DDE_INITIATE

This message is sent by an application to one or more other applications, to request initiation of a conversation.

This message is always sent.

Parameters

param1

hwnd *(HWND)*

Sender's window handle.

param2

data *(PDDEINIT)*

Pointer to initiation data.

This points to a *DDEINIT* structure. **appname** is the name of the desired server application; if this is a zero-length string, any application can respond. **topic** is the name of the desired topic; if this is a zero-length string, each responding application responds once for each topic that it can support.

Returns

reply

result *(BOOL)*

Success indicator:

TRUE Successful completion
FALSE Error occurred.

Remarks

Upon receiving this message, all applications with names matching the application name (where specified), that support the topic identified by the topic name, are expected to acknowledge by calling WinDdeRespond.

Default Processing

None.

WM_DDE_INITIATEACK

This message is sent by a server application in response to a WM_DDE_INITIATE message, for each topic that the server application wishes to support. The application must use WinDdeRespond to send this message.

This message is always sent.

Parameters

param1

hwnd *(HWND)*

Sender's window handle.

param2

data *(PDDEINIT)*
Pointer to initiation data.

This points to a *DDEINIT* structure. **appname** is the name of the responding server application; it must not be a zero-length string. **topic** is the name of the topic that the server is willing to support; it must not be a zero-length string.

The *DDEINIT* structure must be in a shareable segment; it is the responsibility of the receiving window procedure to free this segment.

Returns

reply

result *(BOOL)*
Success indicator:

TRUE	Successful completion
FALSE	Error occurred.

Default Processing

None.

WM_DDE_POKE

This message requests an application to accept an unsolicited data item. It is always posted.

Parameters

param1

hwnd *(HWND)*
Sender's window handle.

param2

ddestruct *(PDDESTRUCT)*
DDE structure.

This points to a dynamic data exchange structure. See DDESTRUCT on page 2-6.

itemname identifies the data item to the receiving application.

data is the data. The format of the data is a registered DDE data format, identified by the **format** field.

Returns

reply *(BIT32)*
Reserved.

NULL	Reserved value.

Remarks

The receiving application is expected to reply with a positive WM_DDE_ACK message if it accepts the unsolicited data, or with a negative WM_DDE_ACK if it does not.

Default Processing

None.

WM_DDE_REQUEST

This message is posted from client to server, to request that the server provide a data item to the client.

This message is always posted.

Parameters

param1

hwnd *(HWND)*
Sender's window handle.

param2

ddestruct *(PDDESTRUCT)*
DDE structure.

This points to a dynamic data exchange structure. See DDESTRUCT on page 2-6.

itemname identifies which data item is being requested.

format identifies in which registered DDE data format the data item is to be rendered.

Returns

reply *(BIT32)*
Reserved.

NULL Reserved value.

Remarks

The receiving application is expected to respond with a WM_DDE_DATA message, containing the requested data, if possible. Otherwise, it is expected to respond with a negative WM_DDE_ACK message.

Default Processing

None.

WM_DDE_TERMINATE

This message is posted by either application participating in a DDE conversation, to terminate that conversation.

This message is always posted.

Parameters

param1

hwnd *(HWND)*
Sender's window handle.

param2

reserved *(BIT32)*
Reserved.

NULL Reserved value.

Returns

reply *(BIT32)*
Reserved.

NULL Reserved value.

Remarks

Upon receiving this message, an application is expected to post a WM_DDE_TERMINATE message in response.

Default Processing

None.

WM_DDE_UNADVISE

This message is posted by a client application to a server application to indicate that the specified item should no longer be updated.

This message is always posted.

Parameters

param1

hwnd *(HWND)*
Sender's window handle.

param2

ddestruct *(PDDESTRUCT)*
DDE structure.

This points to a dynamic data exchange structure. See DDESTRUCT on page 2-6. **Itemname** identifies which data update request is to be retracted. If this is a zero-length string, data update requests for all items are retracted.

Returns

reply *(BIT32)*
Reserved.

NULL Reserved value.

Remarks

The receiving application is expected to reply with a positive WM_DDE_ACK message if it can honor the request, or a negative one if it cannot.

Default Processing

None.

Chapter 25. Help Manager Messages

HM_ACTION_BAR_COMMAND

This message is sent to the current active application window by the help manager to notify the application when the user selects a tailored action bar item.

Parameters

param1

command *(IDENTITY)*
Identity of the action bar item that was selected.

reserved *(BIT16)*
Reserved.

NULL Reserved value.

param2 *(BIT32)*
Reserved.

NULL Reserved value.

Returns

reply *(BIT32)*
Reserved.

NULL Reserved value.

Remarks

This message enables the application to perform some application specific function.

Default Processing

None.

HM_CREATE_HELP_TABLE

This message is sent by the application to give the help manager a new help table.

Parameters

param1

helptable *(PHELPTABLE)*
Help table.

This points to a help table structure; see HELPTABLE on page 2-19.

param2 *(BIT32)*
Reserved.

NULL Reserved value.

Returns

reply

returnvalue *(ULONG)*
Return code.

0 The procedure was successfully completed
Other See the values of the **ErrorCode** parameter of the HM_ERROR message.

Default Processing
None.

HM_DISMISS_WINDOW
This message tells the help manager to remove the help window associated with the last active window.

Parameters
param1 *(BIT32)*
Reserved.

NULL Reserved value.

param2 *(BIT32)*
Reserved.

NULL Reserved value.

Returns
reply

returnvalue *(ULONG)*
Return code.

0	The help window was successfully removed
Other	There was no associated help window.

See also the values of the **errorcode** parameter of the HM_ERROR message.

Remarks
If the user requests help from a primary or secondary window, and then interacts with the primary or secondary window without leaving help, the currently displayed help window might not be appropriate for the application window. This message gives the application the ability to remove that help window.

Default Processing
None.

HM_DISPLAY_HELP
This message tells the help manager to display a specific help panel.

Parameters
param1

This parameter depends on the value of the **typeflag** parameter.

For a value of the **typeflag** parameter of HM_RESOURCEID.

helppanelId *(PIDENTITY)*
Identity of the help panel.

For a value of the **typeflag** parameter of HM_PANELNAME.

helppanelname *(PSTRL)*
Name of the help panel.

param2

typeflag *(USHORT)*
Flag indicating how to interpret the first parameter.

HM_RESOURCEID	Indicates the **param1** points to the identity of the help panel
HM_PANELNAME	Indicates the **param1** points to the name of the help panel.

Returns

reply

returnvalue *(ULONG)*
Return code.

0	The panel was successfully displayed
Other	See the values of the **errorcode** parameter of the HM_ERROR message.

Remarks

The help manager assumes that the help panel is to be associated with the currently active application window, given to the help manager with the HM_SET_ACTIVE_WINDOW message. If the HM_SET_ACTIVE_WINDOW message is not used, the help manager uses the active application window returned by the WinQueryActiveWindow call.

Default Processing

None.

HM_ERROR

This message notifies the application of an error caused by a user interaction.

Parameters

param1

errorcode *(ULONG)*
Error code.

A constant describing the type of error that occurred. The application also returns some of these same messages by messages sent to the help manager.

The error constants are:

HMERR_NO_FRAME_WND_IN_CHAIN	There is no frame window in the window chain from which to find or set the associated help instance.
HMERR_INVALID_ASSOC_APP_WND	The application window handle specified on the WinAssociateHelpInstance call is not a valid window handle.
HMERR_INVALID_ASSOC_HELP_INST	The help instance handle specified on the WinAssociateHelpInstance call is not a valid window handle.
HMERR_INVALID_DESTROY_HELP_INST	The window handle specified as the help instance to destroy is not of the help instance class.
HMERR_NO_HELP_INST_IN_CHAIN	The parent or owner chain of the application window specified does not have an associated help instance.
HMERR_INVALID_HELP_INSTANCE_HDL	The handle specified to be a help instance does not have the class name of a help manager help instance.
HMERR_INVALID_QUERY_APP_WND	The application window specified on a WinQueryHelpInstance call is not a valid window handle.
HMERR_HELP_INST_CALLED_INVALID	The handle of the help instance specified on a call to the help manager does not have the class name of a help manager help instance.
HMERR_HELPTABLE_UNDEFINE	
HMERR_HELP_INSTANCE_UNDEFINE	
HMERR_HELPITEM_NOT_FOUND	
HMERR_INVALID_HELPSUIBITEM_SIZE	
HMERR_HELPSUIBITEM_NOT_FOUND	
HMERR_INDEX_NOT_FOUND	The index is not in the library file.
HMERR_CONTENT_NOT_FOUND	The library file does not have any content.
HMERR_OPEN_LIB_FILE	The library file cannot be opened.
HMERR_READ_LIB_FILE	The library file cannot be read.
HMERR_CLOSE_LIB_FILE	The library file cannot be closed.
HMERR_INVALID_LIB_FILE	Improper library file provided.

HMERR_NO_MEMORY	Unable to allocate the requested amount of memory.
HMERR_ALLOCATE_SEGMENT	Unable to allocate a segment of memory for memory allocation requests from the help manager.
HMERR_FREE_MEMORY	Unable to free allocated memory.
HMERR_PANEL_NOT_FOUND	Unable to find the requested help panel.
HMERR_DATABASE_NOT_OPEN	Unable to read the unopened database.

param2 *(BIT32)*
Reserved.

NULL Reserved value.

Returns

reply *(BIT32)*
Reserved.

NULL Reserved value.

Remarks

There is no other way to communicate the error to the application since the user initiated communication, not the application. Other errors caused when the application sends a message to the help manager are returned as the **reply** parameter of the message.

The help manager does not display any error messages to the user. Instead, the help manager sends or returns all error notifications to the application so that it can display its own messages. This procedure ensures a consistent message interface for all user messages.

Default Processing

None.

HM_EXT_HELP

When the help manager receives this message, it displays the extended help panel for the active application panel.

Parameters

param1 *(BIT32)*
Reserved.

NULL Reserved value.

param2 *(BIT32)*
Reserved.

NULL Reserved value.

Returns

reply

returnvalue *(ULONG)*
Return code.

0	The extended help panel was successfully displayed
Other	See the values of the **errorcode** parameter of the HM_ERROR message.

Default Processing

None.

HM_EXT_HELP_UNDEFINED

This message is sent to the application by the help manager to notify it that an extended help panel has not been defined.

Parameters

param1 *(BIT32)*
Reserved.

NULL Reserved value.

param2 *(BIT32)*
Reserved.

NULL Reserved value.

Returns

reply *(BIT32)*
Reserved.

NULL Reserved value.

Remarks

When the extended help panel is requested, the help manager searches the help table for its identity. If the extended help panel identity associated with the current active window is zero, the help manager sends this message to the application to notify it that an extended help panel has not been defined. The application then can:

- Ignore the request for help and not display a help panel
- Display its own panel
- Use the HM_DISPLAY_HELP message to tell the help manager to display a particular panel.

Default Processing

None.

HM_HELP_CONTENTS

When the help manager receives this message, it displays the help contents panel.

Parameters

param1 *(BIT32)*
Reserved.

NULL Reserved value.

param2 *(BIT32)*
Reserved.

NULL Reserved value.

Returns

reply

returnvalue *(ULONG)*
Return code.

0 The help contents panel was successfully displayed
Other See the values of the **errorcode** parameter of the HM_ERROR message.

Default Processing
None.

HM_HELP_INDEX
When the help manager receives this message, it displays the help index panel.

Parameters
param1 *(BIT32)*
Reserved.

NULL Reserved value.

param2 *(BIT32)*
Reserved.

NULL Reserved value.

Returns
reply

returnvalue *(ULONG)*
Return code.

0 The help index panel was successfully displayed
Other See the values of the **errorcode** parameter of the HM_ERROR message.

Default Processing
None.

HM_HELPSUBITEM_NOT_FOUND
The help manager sends this message to the application when the user requests help on a field and it cannot find a related entry in the help subtable.

Parameters
param1

context *(USHORT)*
Type of window on which help was requested.

HLPM_WINDOW An application window
HLPM_FRAME A frame window
HLPM_MENU A menu window.

reserved *(USHORT)*
Reserved.

NULL Reserved value.

param2

topic *(USHORT)*
Topic identifier.

For a value of the **context** parameter of HLPM_WINDOW or HLPM_FRAME:

window Identity of the window containing the field on which help was requested.

For a value of the **context** parameter of HLPM_MENU:

menu Identity of the submenu containing the field on which help was requested.

subtopic *(USHORT)*
Subtopic identifier.

For a value of the **context** parameter of HLPM_WINDOW or HLPM_FRAME:

control Control identity of the cursored field on which help was requested.

For a value of the **context** parameter of HLPM_MENU:

–1 No menu item was selected
Other Menu item identity of the currently selected submenu item on which help was requested.

Returns

reply

Informs the help manager what should be done next.

action *(BOOL)*
Action indicator:

For a value of the **context** parameter of HLPM_WINDOW or HLPM_FRAME:

FALSE Display the extended help panel
TRUE Do nothing.

For a value of the **context** parameter of HLPM_MENU:

FALSE Display the extended help panel.

Remarks

If FALSE is returned from this message, the help manager displays the extended help panel.

The application has the following options:

- Ignore the notification and not display help for that field or panel
- Display its own panel
- Use the HM_DISPLAY_HELP message to tell the help manager to display a particular panel.

Default Processing

None.

HM_INFORM

This message is used by the help manager to notify the application when the user selects a hypertext field that was specified with the inform tag.

Parameters

param1

num *(IDENTITY)*
Window identity.

The identity that is associated with the hypertext field.

param2 *(BIT32)*
Reserved.

NULL Reserved value.

Returns

reply *(BIT32)*
Reserved.

NULL Reserved value.

Default Processing

None.

HM_KEYS_HELP

Parameters

param1 *(BIT32)*
Reserved.

NULL Reserved value.

param2 *(BIT32)*
Reserved.

NULL Reserved value.

Returns

reply

returnvalue *(ULONG)*
Return code.

0 The keys help panel was successfully displayed
Other See the values of the **errorcode** parameter of the HM_ERROR message.

Remarks

When the help manager receives this message, it sends a HM_QUERY_KEYS_HELP message to the active application window. The active application window which receives the HM_QUERY_KEYS_HELP message is the window specified when the last HM_SET_ACTIVE_WINDOW message was sent. If no HM_SET_ACTIVE_WINDOW message was issued, it is the application window specified in the WinAssociateHelpInstance call. The application must return the identity of a keys help panel in the **HelpPanel** parameter of the HM_QUERY_KEYS_HELP message or zero, if no action is to be taken by the help manager for keys help.

Default Processing

None.

HM_LOAD_HELP_TABLE

The application sends this message to give the help manager the module handle that contains the help table and help subtable and the identity of the help table.

Parameters

param1

helptable *(IDENTITY)*
Identity of the help table.

X'ffff' *(USHORT)*

param2

module *(RESID)*
Handle of the module that contains the help table and help subtable.

reserved *(BIT32)*
Reserved.

NULL Reserved value.

Returns

reply

returnvalue *(ULONG)*
Return code.

0 The procedure was successfully completed
Other See the values of the **errorcode** parameter of the HM_ERROR message.

Default Processing
None.

HM_QUERY_KEYS_HELP
When the user requests the keys help function, the help manager sends this message to the application.

Parameters
param1 *(BIT32)*
Reserved.

NULL Reserved value.

param2 *(BIT32)*
Reserved.

NULL Reserved value.

Returns
reply

helppanel *(USHORT)*
The identity of the application-defined keys help panel to be displayed.

0 Do nothing
Other Identity of the keys help panel to be displayed.

Remarks
The application responds by returning the identity of the requested keys help panel. The help manager then displays that help panel. Returning zero in the **helppanel** parameter indicates that the help manager should do nothing for the keys help function.

Default Processing
None.

HM_REPLACE_HELP_FOR_HELP
This message tells the help manager to display the application-defined help for help instead of the help manager help for help panel.

Parameters
param1

helpforhelppanel *(IDENTITY)*
Identity of the application-defined help for help panel.

0 Use the help manager help for help panel
Other Identity of the application-defined help for help panel.

reserved *(BIT16)*
Reserved.

NULL Reserved value.

param2 *(BIT32)*
Reserved.

NULL Reserved value.

Returns

reply *(BIT32)*

Reserved.

NULL Reserved value.

Remarks

An application may prefer to provide information that is more specific to itself, rather than the more general help information provided in the help manager help for help panel.

Default Processing

None.

HM_SET_ACTIVE_WINDOW

This message allows the application to change the window with which the help manager communicates and the window to which the help window is to be positioned.

Parameters

param1

activewindow *(HWND)*

Handle of the window to be made active.

Its window procedure receives all messages from the help manager until the application changes the active window with another HM_SET_ACTIVE_WINDOW message.

param2

relativewindow *(HWND)*

Handle of the window next to which the help window is to be positioned.

The handle of the application window next to which the help manager will position a new help window.

HWND_PARENT	This help manager-defined constant tells the help manager to trace the parent chain of the window that had the focus when the user requested help.
Other	Handle of the window next to which the help window is to be positioned.

Returns

reply

returnvalue *(ULONG)*

Return code.

0	The procedure has been successfully completed
Other	See the values of the **errorcode** parameter of the HM_ERROR message.

Remarks

Normally the help manager communicates with the application window with which the help manager instance has been associated. The help window is positioned next to this same application window.

If the **activewindow** parameter is zero, the **relativewindow** parameter is set to zero, that is, if the active window is NULL, the relative window is not used.

Default Processing

None.

HM_SET_HELP_LIBRARY_NAME

This message identifies a list of help panel library names to the help manager instance.

Parameters

param1

helplibraryname *(PSTRL)*
Library name.

This points to a *STRL* data type.

The string contains a list of help panel library names that will be searched by the help manager for the requested help panel.

param2 *(BIT32)*
Reserved.

NULL Reserved value.

Returns

reply

returnvalue *(ULONG)*
Return code.

0 The new specified library successfully replaced the current help panel library name.
Other See the values of the **errorcode** parameter of the HM_ERROR message.

Remarks

Any subsequent communication to the help manager with this message replaces the current list of names with the newly specified list.

When help is requested, the help manager will search each library in the list for the requested help panel.

Default Processing

None.

HM_SET_HELP_WINDOW_TITLE

This message allows the application to change the window text of a help window title.

Parameters

param1

helpwindowtitle *(PSTRL)*
Help window title.

This points to a *STRL* data type.

param2 *(BIT32)*
Reserved.

NULL Reserved value.

Returns

reply

returnvalue *(ULONG)*
Return code.

0 The window title was successfully set
Other See the values of the **errorcode** parameter of the HM_ERROR message.

Default Processing
None.

HM_SET_SHOW_PANEL_ID
This message tells the help manager to display, hide, or toggle the panel identity for each help panel displayed.

Parameters
param1

showpanelid *(BIT16)*
Show panel identity indicator:

CMIC_HIDE_PANEL_ID	Sets the show option off and the panel identity is not displayed
CMIC_SHOW_PANEL_ID	Sets the show option on and the panel identity is displayed
CMIC_TOGGLE_PANEL_ID	Toggles the display of the panel identity.

reserved *(BIT16)*
Reserved.

NULL Reserved value.

param2 *(BIT32)*
Reserved.

NULL Reserved value.

Returns
reply

returnvalue *(ULONG)*
Return code.

0	The show panel identity indicator was successfully changed
Other	See the values of the **errorcode** parameter of the HM_ERROR message.

Default Processing
None.

HM_TUTORIAL
The help manager sends this message to the application window when the user selects the tutorial option from a help panel.

Parameters
param1

tutorialname *(PSTRL)*
Default tutorial name.

This points to a *STRL* data type.

The string contains the name of the default tutorial program specified in the help manager initialization structure. A tutorial name specified in the help panel definition overrides this default tutorial program.

param2 *(BIT32)*
Reserved.

NULL Reserved value.

Returns

reply *(BIT32)*

Reserved.

NULL Reserved value.

Remarks

The application then calls its own tutorial program.

Default Processing

None.

Chapter 26. Resource Files

This chapter describes the syntax for the resource language using railroad syntax, and describes the formats used.

Resource files are used to build dialog templates, menu templates, accelerator tables, extended attribute association tables, keyboard scancode mapping tables, keyboard names and fonts. The files must be compiled before they can be used by application programs.

How to Read the Syntax Definitions

Throughout this book, syntax is described using the structure defined below.

- Read the syntax diagrams from left to right, from top to bottom, following the path of the line.

 The ▶▶— symbol indicates the beginning of a statement.

 The —▶ symbol indicates that the statement syntax is continued on the next line.

 The ▶— symbol indicates that a statement is continued from the previous line.

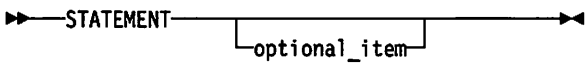
 The —▶◀ symbol indicates the end of a statement.

 Diagrams of syntactical units other than complete statements start with the ▶— symbol and end with the —▶ symbol.

- Required items appear on the horizontal line (the main path).

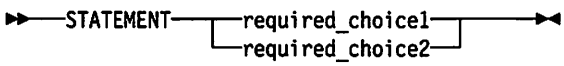


- Optional items appear below the main path.

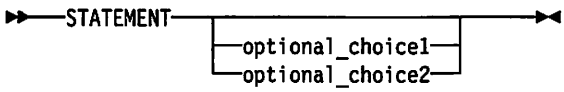


- If a choice can be made from two or more items, they appear vertically, in a stack.

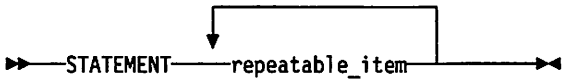
 If one of the items *must* be chosen, one item of the stack appears on the main path.



 If choosing one of the items is optional, the entire stack appears below the main path.



- An arrow returning to the left above the main path indicates an item that can be repeated.



 A repeat arrow above a stack indicates that a choice can be made from the stacked items, or a single choice can be repeated.

- Keywords appear in uppercase (for example, PARM1). They must be spelled exactly as shown. Variables appear in all lowercase letters (for example, parmx). They represent user-supplied names or values.
- If punctuation marks, parentheses, arithmetic operators, or such symbols are shown, they must be entered as part of the syntax.

Definitions Used in all Resources

Specification of Values

These rules apply to values specified in resources:

Coordinates must be integers. There must be no space between the sign of the value and the value itself. For example, "–1" is allowed but "– 1" is not.

Resource identifiers must be positive integers or names that resolve positive integers.

Real values, containing a decimal point, cannot be used.

Resource Load and Memory Options

The following options define when each resource is loaded and how memory is allocated for each resource.

LOADOPTION	Resource loading options	
	PRELOAD	Resource is loaded immediately.
	LOADONCALL	Resource is loaded when called.
MEMOPTION	Resource memory options	
	FIXED	Resource remains at a fixed memory location.
	MOVEABLE	Resource can be moved if necessary to compact memory.
	DISCARDABLE	Resource can be discarded if no longer needed.

Resource Script File Specification

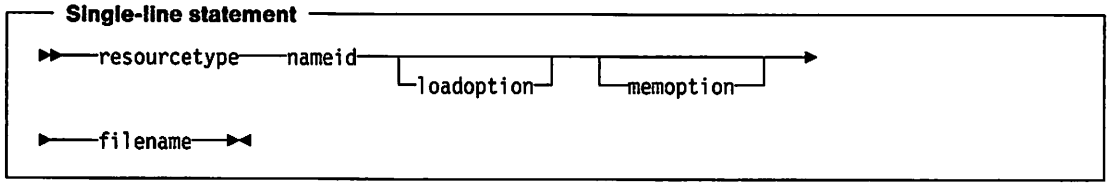
The resource script file defines the names and attributes of the resources to be added to the application's executable file. The file consists of one or more resource statements that define the resource type and original file, if any. The following is a list of the types of resource statement:

- Single-line statements
- User-defined resources
- Directives
- Multiple-line statements.

The following sections describe these statements in detail.

Single-Line Statements

The general form for all single-line statements is:



resourcetype *(USHORT)*
is one of the following keywords, specifying the type of resource to be loaded:

Keyword	Resource type
FONT	A font resource is simply a file containing a font.
POINTER	A pointer resource is a bit map defining the shape of the pointing device pointer on the display screen.
ICON	An icon resource is a bit map defining the shape of the icon to be used for a given application.
BITMAP	A bit-map resource is a custom bit map that an application intends to use in its screen display or as an item in a menu.
DLGINCLUDE	This statement tells the dialog editor which file to use as an include file for the dialogs in the resource file. The **nameid** is not applicable.

nameid *(USHORT)*
is either a unique name or an integer number identifying the resource. For a FONT resource, the **nameid** must be a number; it cannot be a name.

loadoption *(LOADOPTION)*
The default is LOADONCALL.

memoption *(MEMOPTION)*
The default is MOVEABLE and DISCARDABLE for POINTER, ICON, and FONT resources. The default for BITMAP resources is MOVEABLE. The FIXED option overrides both MOVEABLE and DISCARDABLE.

filename *(STR)*
is an ASCII string specifying the OS/2 Version 1.2 name of the file containing the resource. A full path name must be given if the file is not in the current working directory.

Example

```
POINTER pointer point.cur
POINTER pointer DISCARDABLE point.cur
POINTER 10 custom.cur

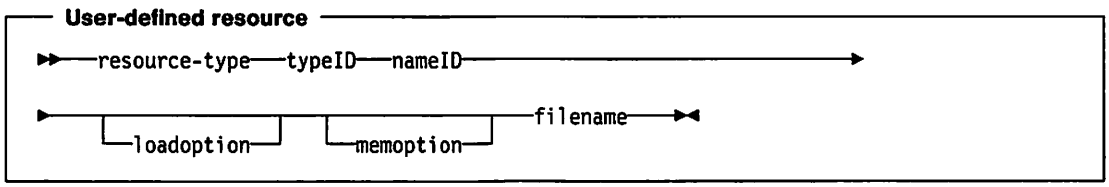
ICON desk desk.ico
ICON desk DISCARDABLE desk.ico
ICON 11 custom.ico

BITMAP disk disk.bmp
BITMAP disk DISCARDABLE disk.bmp
BITMAP 12 custom.bmp

FONT 5 CMROMAN.FNT
```

User-Defined Resources

An application can also define its own resource. The resource can be any data that the application intends to use. A user-defined resource statement has the form:



typeID
is either a unique name or an integer number identifying the resource type. If a number is given, it must be greater than 255. The type numbers 1 through 255 are reserved for existing and future predefined resource types.

nameID
is either a unique name or an integer number identifying the resource.

loadoption *(LOADOPTION)*
The default is LOADONCALL.

memoption *(MEMOPTION)*
The default is MOVEABLE.

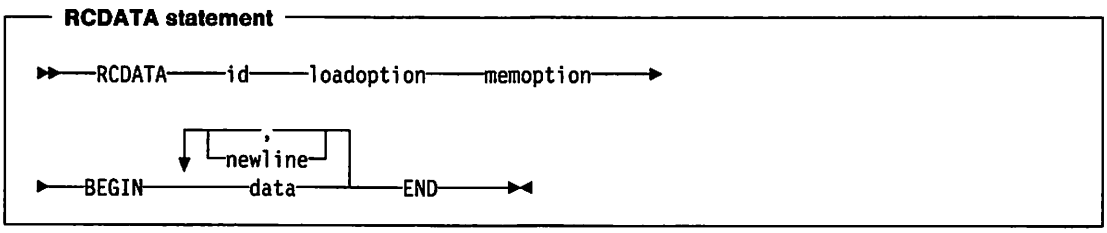
filename
is an ASCII string specifying the OS/2 Version 1.2 name of the file containing the cursor bit map. A full path name must be given if the file is not in the current working directory.

Example

```
RESOURCE MYRES   array   DATA.RES
RESOURCE 300     14      CUSTOM.RES
```

RCDATA statement

The RCDATA statement is provided to allow an application to define a simple data resource.



id
is either a unique name or an integer number identifying the resource.

loadoption *(LOADOPTION)*
The default is LOADONCALL.

memoption *(MEMOPTION)*
The default is MOVEABLE.

data
is a number or string.

Example

```
RCDATA 4
BEGIN
 "Sample string."
 "TEST DATA."
 "A message."
END
```

Directives

The resource directives are special statements that define actions to perform on the file before it is compiled. The directives can assign values to names, include the contents of files, and control compilation of the file.

#include filename

#rcinclude filename

These directives copy the contents of the file specified by **filename** into the resource before it is compiled. If **#rcinclude** is used, the entire file is copied. If **#include** is used, only **#define** statements are copied.

Programming Note: If an **#rcinclude** is to be commented out, the open comment (/*) must appear on the same line as the directive.

Filename is an ASCII string. A full path name must be given if the file is not in the current directory or in the directory specified by the INCLUDE environment variable. The file extensions .I and .TMP must not be used as these are reserved for system use.

The **filename** parameter is handled as a C string, and two back-slashes must be given wherever one is expected in the path name (for example, root\\sub.) Or, a single forward slash (/) can be used instead of double back-slashes (for example, root/sub.).

Example

```
#include "wincalls.h"

MENU PenSelect
BEGIN
       MENUITEM "black pen", BLACK_PEN
END
```

Files included in resource script files constants that use #define statements may not include any casting of those constants that are used in the resource script. The resource compiler does not parse this casting syntax. For example, the following statement may not be included:

```
#define IDBUTTON1  (USHORT) 3
```

If casting is required for C source compilation, you may use two statements such as:

```
#define IDBUTTON1    3
#define CSRC_IDBUTTON1    ((USHORT)IDBUTTON1)
```

#define name value

This directive assigns the given value to **name**. All subsequent occurrences of **name** are replaced by the value.

name is any combination of letters, digits, or punctuation.

value is any integer, character string, or line of text.

Example

```
#define     nonzero        1
#define     USERCLASS   "MyControlClass"
```

#undef name

This directive removes the current definition of **name**. All subsequent occurrences of **name** are processed without replacement.

name is any combination of letters, digits, or punctuation.

Example

```
#undef      nonzero
#undef      USERCLASS
```

#ifdef name

This directive performs a conditional compilation of the resource file by checking the specified name. If the name has been defined using a #define directive, #ifdef directs the resource compiler to continue with the statement immediately after it. If the name has not been defined, #ifdef directs the compiler to skip all statements up to the next #endif directive.

name is the name to be checked by the directive.

Example

```
#ifdef Debug
FONT 4 errfont.fnt
#endif
```

#ifndef name

This directive performs a conditional compilation of the resource file by checking the specified name. If the name has not been defined or if its definition has been removed using the #undef

directive, #ifndef directs the resource compiler to continue processing statements up to the next #endif, #else, or #elif directive, then skip to the statement after the #endif. If the name is defined, #ifndef directs the compiler to skip to the next #endif, #else, or #elif directive.

name is the name to be checked by the directive.

Example

```
#ifndef Optimize
FONT 4 errfont.fnt
#endif
```

#if constant expression

This directive performs a conditional compilation of the resource file by checking the specified constant-expression. If the constant-expression is nonzero, #if directs the resource compiler to continue processing statements up to the next #endif, #else, or #elif directive, then skip to the statement after the #endif. If the constant-expression is zero, #if directs the compiler to skip to the next #endif, #else, or #elif directive.

constant expression is a defined name, an integer constant, or an expression consisting of names, integers, and arithmetic and relational operators.

Example

```
#if Version<3
FONT 4 errfont.fnt
#endif
```

#elif constant expression

This directive marks an optional clause of a conditional compilation block defined by an #ifdef, #ifndef, or #if directive. The directive carries out conditional compilation of the resource file by checking the specified constant-expression. If the constant-expression is nonzero, #elif directs the resource compiler to continue processing statements up to the next #endif, #else, or #elif directive, then skip to the statement after the #endif. If the constant-expression is zero, #elif directs the compiler to skip to the next #endif, #else, or #elif directive. Any number of #elif directives can be used in a conditional block.

constant expression is a defined name, an integer constant, or an expression consisting of names, integers, and arithmetic and relational operators.

Example

```
#if Version<3
FONT 4 italic.fnt
#elif Version<7
FONT 4 bold.fnt
#endif
```

#else

This directive marks an optional clause of a conditional compilation block defined by an #ifdef, #ifndef, or #if directive. The #else directive must be the last directive before #endif.

Example

```
#ifdef Debug
FONT 4 italic.fnt
#else
FONT 4 bold.fnt
#endif
```

#endif

This directive marks the end of a conditional compilation block defined by an #ifdef, #ifndef, or #if directive. One #endif is required for each #ifdef, #ifndef, and #if directive.

Multiple-Line Statements

Code Page Flagging

The CODEPAGE statement may be placed within the source, to set the code page used for these resources:

STRINGTABLE
ACCELTABLE
MENU
DIALOGTEMPLATE and WINDOWTEMPLATE.

The CODEPAGE statement cannot be encoded within any other statement. All items following a CODEPAGE statement are assumed to be in that code page. The code page is encoded in the resource, and the data in the resource is assumed to be in the specified code page. However, no checking is performed.

These code pages can be specified:

437
850
860
863
865.

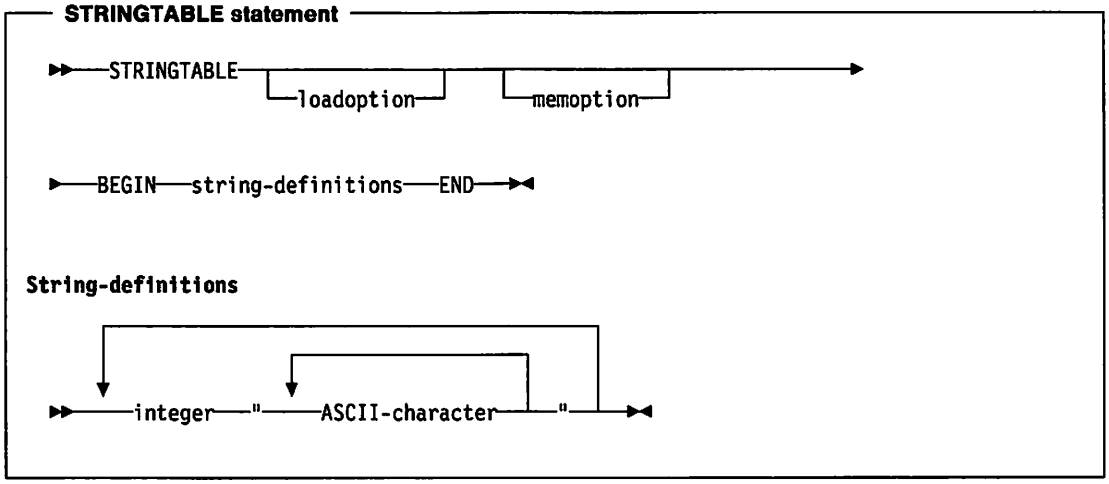
If the code page is not specified, code page 850 is assumed.

STRINGTABLE Statement

The STRINGTABLE statement defines one or more string resources for an application. String resources are simply null-terminated ASCII strings that can be loaded, when needed, from the executable file, using the WinLoadString function.

Programming Note: The ASCII strings can include no more than 256 characters, including the NULL termination character.

The STRINGTABLE statement has the form:



loadoption *(LDOPT)*
is an optional keyword specifying when the resource is to be loaded. It must be one of:

PRELOAD	Resource is loaded immediately
LOADONCALL	Resource is loaded when called.

The default is LOADONCALL.

memoption *(MEMOPT)*
consists of the following keyword or keywords, specifying whether the resource is fixed or moveable and whether it is discardable:

FIXED	Resource remains at a fixed memory location
MOVEABLE	Resource can be moved if necessary to compact memory
DISCARDABLE	Resource can be discarded if no longer needed.

The default is MOVEABLE and DISCARDABLE.

ASCII-character
This may be any ASCII character. Since '\' is interpreted as an escape character, use '\\' to generate a '\'.

The following escape sequences may be used:

Escape Sequence	**Name**
\t	Horizontal tab
\a	Bell (alert)
\nnn	ASCII character (octal)
\xdd	ASCII character (hexadecimal).

The sequences \ddd and \xdd allow any character in the ASCII character set to be inserted in the character string. Thus, the horizontal tab could be entered as \X09, \011 or \t.

Example

```
#define IDS_HELLO    1
#define IDS_GOODBYE  2

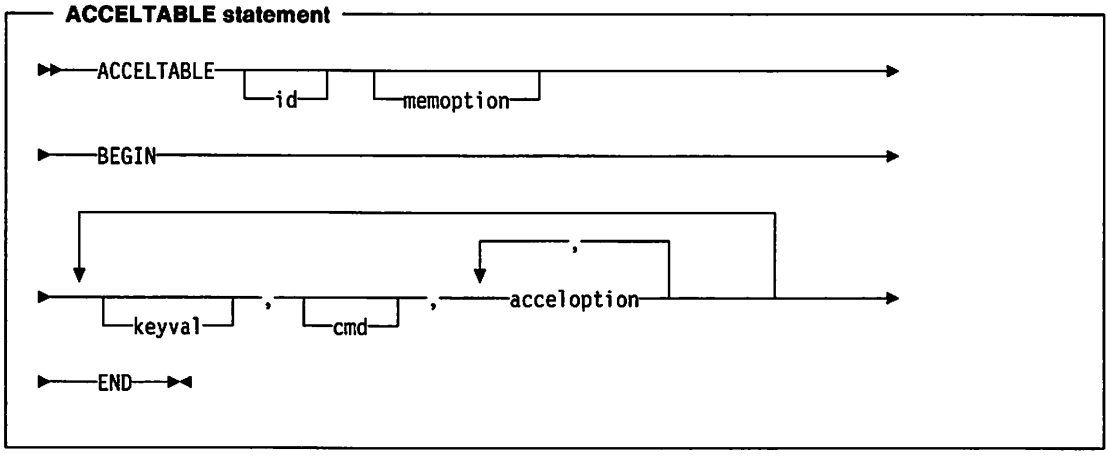
STRINGTABLE
BEGIN
    IDS_HELLO,   "Hello"
    IDS_GOODBYE, "Goodbye"
END
```

ACCELTABLE Statement

The ACCELTABLE statement defines a table of *accelerator* keys for an application.

An accelerator is a key stroke defined by the application to give the user a quick way to perform a task. The WinGetMsg call automatically translates accelerator messages from the application queue into WM_COMMAND, WM_HELP, or WM_SYSCOMMAND messages.

The ACCELTABLE statement has the form:



id *(USHORT)*
is the resource identifier.

memoption
is optional. It consists of the following keyword or keywords, specifying whether the resource is fixed or moveable, and whether it can be discarded:

FIXED	Resource remains at a fixed memory location.
MOVEABLE	Resource can be moved if necessary to compact memory.
DISCARDABLE	Resource can be discarded if no longer needed.

keyval *(USHORT)*
is the accelerator character code. This can be either a constant or a quoted character. If it is a quoted character, the CHAR acceloption is assumed. If the quoted character is preceded with a caret character (^), a control character is specified as if the CONTROL **acceloption** had been used.

cmd *(USHORT)*
is the value of the WM_COMMAND, WM_HELP, or WM_SYSCOMMAND message generated from the accelerator for the indicated key.

acceloption *(BIT_16)*
defines the kind of accelerator.

These options are available:

ALT
CHAR
CONTROL
HELP
LONEKEY
SCANCODE
SHIFT
SYSCOMMAND
VIRTUALKEY.

The VIRTUALKEY, SCANCODE, LONEKEY, and CHAR acceloptions specify the type of message that matches the accelerator. Only one of these options can be specified for each accelerator. For information on the corresponding KC_* values, see "WM_CHAR" on page 12-14.

The **acceloptions** SHIFT, CONTROL, and ALT, cause a match of the accelerator only if the corresponding key is down.

If there are two accelerators that use the same key with different SHIFT, CONTROL, or ALT options, the more restrictive accelerator should be specified first in the table. For example, Shift-Enter should be placed before Enter.

The SYSCOMMAND **acceloption** causes the key stroke to be passed to the application as a WM_SYSCOMMAND message. The HELP **acceloption** causes the key stroke to be passed to the application as a WM_HELP message. If neither is specified, a WM_COMMAND message is used.

Example

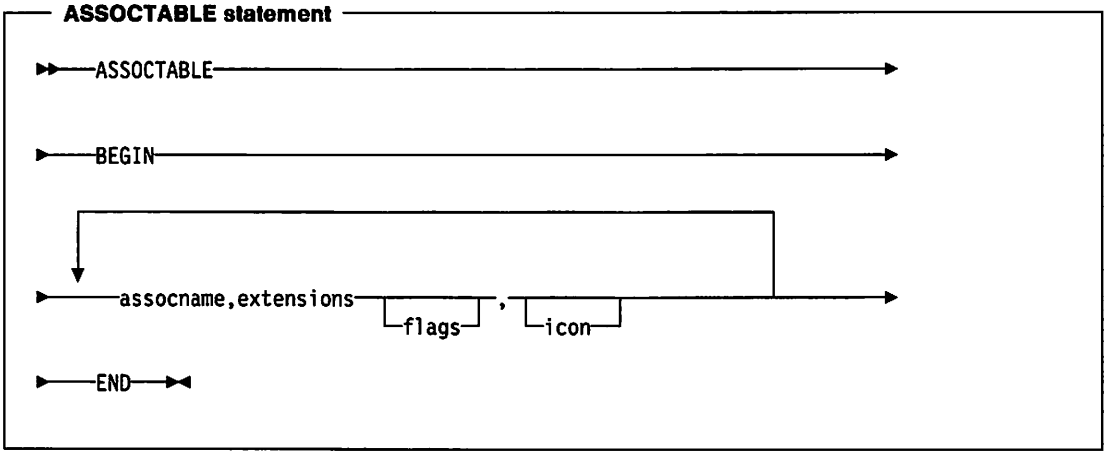
```
ACCELTABLE MainAcc
BEGIN
     VK_F1,101,HELP
     VK_F3,102,SYSCOMMAND
END
```

This generates a WM_HELP with value 101 from VIRTUALKEY accelerator F1 and a WM_SYSCOMMAND with value 102 from VIRTUALKEY accelerator F3.

ASSOCTABLE Statement

The ASSOCTABLE statement defines the extended attributes (EA) for an application.

The ASSOCTABLE statement has the form:



The source for the ASSOCTABLE description is contained in the resource file for a particular project:

```
ASSOCTABLE
BEGIN
    "association name", "extension", flags, icon filename
    "association name", "extension", flags, icon filename
    ...
END
```

Association name Program recognizes data files of this EA TYPE.

Extension 3 letter file extension that is used to identify files of this type if they have no EA TYPE entry. (This may be empty.)

flags

AF_DEFAULTOWNER.
The default application for the file.

AF_UNCHANGEABLE
This flag is set if the entry in the ASSOCTABLE is not to be edited.

AF_REUSEICON
This flag is specified if a previously defined icon in the ASSOCTABLE is to be reused. Entries with this flag set have no icon data defined. The icon used for this entry is the icon used for the previous entry. For example:

```
ASSOCTABLE
BEGIN
    "Product XYZ Spreadsheet", "xys", AF_DEFAULTOWNER, xyzspr.ico
    "Product XYZ Chart", "xyc", AF_DEFAULTOWNER | AF_REUSEICON
END
```

Note that AF_* flags may be ORed together when specified in the ASSOCTABLE.

Icon filename Filename of the icon used to represent this file type. (This may be empty.)

DEFAULTICON Keyword

This keyword installs the filename.ico icon definition under the ICON EA of the program file.

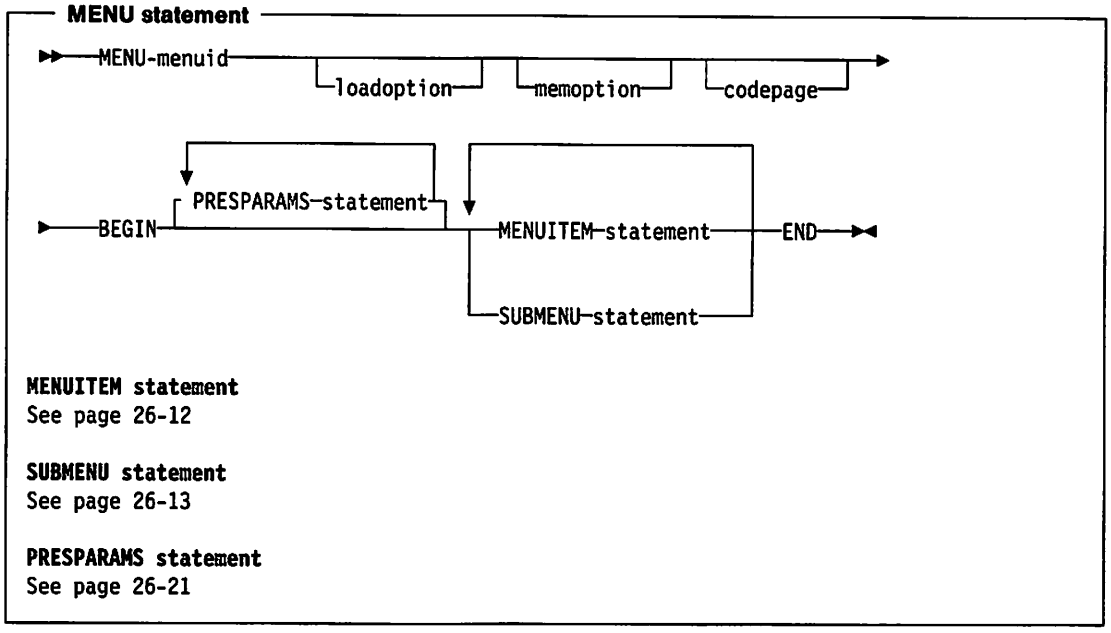
Example

```
DEFAULTICON <filename.ico>
```

MENU Statement

The MENU statement defines the contents of a menu resource. A menu resource is a collection of information that defines the appearance and function of an application menu. A menu can be used to create an action bar.

The MENU statement has the form:



menuid *(USHORT)*
is a name or number used to identify the menu resource.

loadoption *(LOADOPTION)*
The default is LOADONCALL.

memoption *(MEMOPTION)*
The default is MOVEABLE.

codepage *(USHORT)*
is the code page of the text.

MENUITEM
is a special resource statement used to define the items in the menu. These are discussed in more detail on page 26-12.

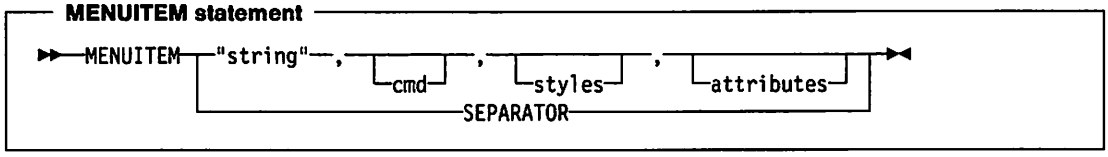
Example: This is an example of a complete MENU statement:

```
MENU sample
BEGIN
   MENUITEM "~Alpha", 100, MIS_TEXT
   SUBMENU  "~Beta", 101, MIS_TEXT
   BEGIN
          MENUITEM "~Green", 200, MIS_TEXT
          MENUITEM "~Blue", 201, MIS_TEXT,MIA_CHECKED
   END
END
```

Menu Item Definition Statements

MENUITEM statements are used in the item-definition section of a MENU statement to define the names and attributes of the actual menu items. Any number of statements can be given; each defines a unique item. The order of the statements defines the order of the menu items.

Programming Note: The MENUITEM statements can only be used within an item-definition section of a MENU statement.



string *(STR)*
is a string, enclosed in double quotation marks, specifying the text of the menu item.

To insert a double-quote character (") in the text, use two double-quote characters ("").

If the **styles** parameter does not contain MIS_TEXT, the string is ignored but must still be specified. An empty string ("") should be specified in this instance.

To indicate the mnemonic for each item, insert the tilde character (~) in the string preceding the mnemonic character.

For MENUITEM statements within a SUBMENU (that is, pull-down menus) text may be split into a second column with an alignment substring. To right-align items insert "\a" in the text where alignment should begin. To left-align a second column of text insert "\t" in the text where alignment should begin. For each SUBMENU the longest item in the second column determines the width of that column. Only one alignment substring should be used in a menu item.

cmd *(USHORT)*
is the value of the WM_COMMAND, WM_HELP, or WM_SYSCOMMAND message generated by the item when it is selected. It identifies the selection made and should be unique within one menu definition.

styles *(BIT_16)*
are one or more menu options defined by the MIS_* constants, ORed together with the | operator. For definitions of the MIS_* constants, see "Menu Item Styles" on page 17-2.

attributes *(BIT_16)*
are one or more menu options defined by the MIA_* constants, ORed together with the | operator. For definitions of the MIA_* constants, see page 17-2.

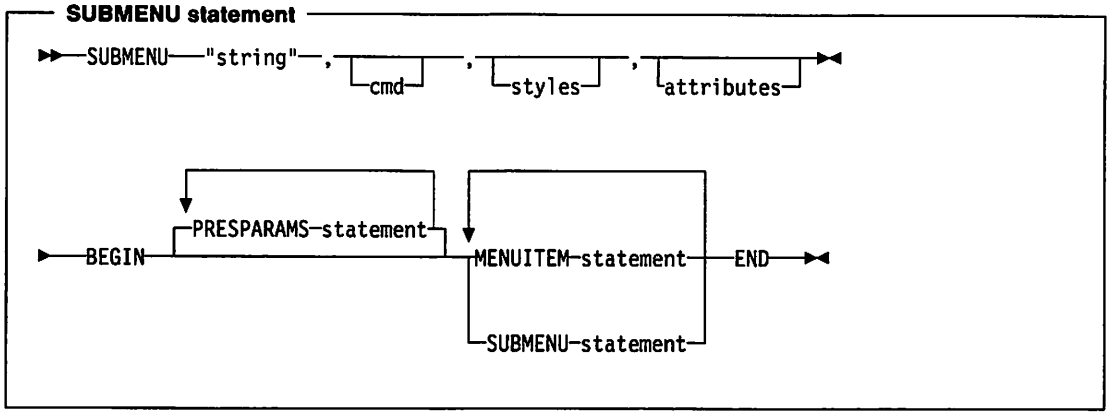
The style MIS_SUBMENU must not be used with this statement. See the SUBMENU statement on page 26-13.

Examples

```
MENUITEM  "Alpha", 1, MIS_TEXT,MIA_ENABLED|MIA_CHECKED,'A'
MENUITEM  "Beta", 2, MIS_TEXT,,'B'
```

Pull-Down Menus or Submenus

In addition to simple items, a menu definition can contain the definition of a submenu. A submenu can itself invoke a lower level submenu.



string *(STR)*
is a string, enclosed in double quotation marks, specifying the text of the menu item.

To insert a double-quote character (") in the text, use two double-quote characters ("").

If the **styles** parameter does not contain MIS_TEXT, the string is ignored but must still be specified. An empty string ("") should be specified in this instance.

cmd *(USHORT)*
is the value of the WM_COMMAND, WM_HELP, or WM_SYSCOMMAND message generated by the item when it is selected. It identifies the selection made and should be unique within one menu definition.

styles *(BIT_16)*
are one or more menu options defined by the MIS_ constants, ORed together with the | operator.

In the SUBMENU statement, the style MIS_SUBMENU is always ORed with the styles given. If no value is supplied, the default value of MIS_TEXT and MIS_SUBMENU is used.

attributes *(BIT_16)*
are one or more menu options defined by the MIA_ constants, ORed together with the | operator.

Example

```
MENU chem
BEGIN

SUBMENU  "~Elements", 2, MIS_TEXT
BEGIN
        MENUITEM "~Oxygen", 200, MIS_TEXT
        MENUITEM "~Carbon", 201, MIS_TEXT,MIA_CHECKED
        MENUITEM "~Hydrogen", 202, MIS_TEXT
END

SUBMENU  "~Compounds", 3, MIS_TEXT
BEGIN
        MENUITEM "~Glucose", 301, MIS_TEXT
        MENUITEM "~Sucrose", 302, MIS_TEXT,MIA_CHECKED
        MENUITEM "~Lactose", 303, MIS_TEXT|MIS_BREAK
        MENUITEM "~Fructose", 304, MIS_TEXT
END

END
```

SEPARATOR Menu Item

There is a special form of the MENUITEM statement that is used to create a horizontal dividing bar between two active menu items in a pull-down menu. The SEPARATOR menu item is itself inactive and has no text associated with it nor a **cmd** value.

Example

```
MENUITEM  "~Roman", 206, MIS_TEXT
MENUITEM  SEPARATOR
MENUITEM  "20 ~Point", 301, MIS_TEXT
```

Menu Template

Menu templates are data structures used to define menus. Menu templates can be loaded as resources or created dynamically, or embedded in dialog templates, which in turn can be loaded as resources or created dynamically. Templates loaded as resources cannot contain references to bit maps or owner-drawn items. A menu template consists of a sequence of variable-length records. Each record in a menu template defines a menu item. If a menu item contains a reference to a submenu, the menu template that defines that submenu is placed after the definition of that particular menu item.

Template Format: A menu template has this format:

Length *(COUNT2B)*
is the length of the menu template.

Version *(USHORT)*
is the template version. The version defined here is version 0.

Code page *(USHORT)*
is the identifier of the code page used for the text items within the menu (but not any submenus, which each have their own code pages).

Item offset *(USHORT)*
is the offset of the items from the start of the template, in bytes.

Count *(COUNT2)*
is the count of menu items.

Reserved *(USHORT)*
is reserved.

Menu items
is a variable-sized array of menu items as follows:

Style *(BIT16)*
Menu item styles (MIS_*; see page 17-2) combined with the logical-OR operator.

Attributes *(BIT16)*
Menu item attributes (MIA_*; see page 17-2) combined with the logical-OR operator.

Item *(IDENTITY)*
An application-provided identifier for the menu item.

Variable data
Following the identifier is a variable data structure whose format depends upon the value of **Style**:

MIS_TEXT

Text *(STRL)*
Null-terminated text string.

MIS_SUBMENU A menu template structure.

MIS_BITMAP

Text *(STR)*
Null-terminated text string.

For MIS_BITMAP menu items, the item text string can be used to derive the resource identifier from which a bit map is loaded. There are three instances:

- The first byte is NULL; that is, no resource is defined and it is assumed that the application subsequently provides a bit-map handle for the item.
- The first byte is X'FF', the second byte is the low byte of the resource identifier, and the third byte is the high byte of the resource identifier.
- The first character is "#," and subsequent characters make up the decimal text representation of the resource identifier.

The resource is assumed to reside in the resource file of the current process.

If the string is empty or does not follow the format above, no resource is loaded.

DLGTEMPLATE and WINDOWTEMPLATE Statements

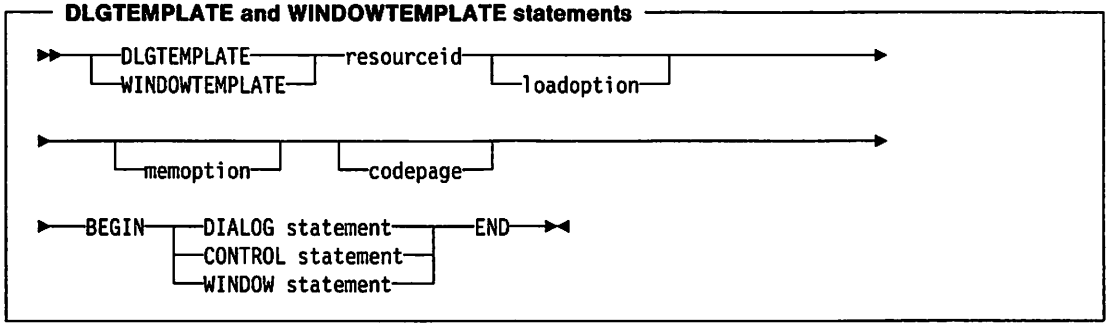
This section describes how to define dialog and menu templates.

It also describes the control data and presentation parameter structures that the application needs to create windows and define dialog templates.

Data types are shown after each parameter or option. These are the data types that the parameter or option is converted to when it is compiled.

DLGTEMPLATE and WINDOWTEMPLATE statements are used by an application to create predefined window and dialog resource templates.

The DLGTEMPLATE and WINDOWTEMPLATE statements are treated identically by the resource compiler and have this format:



The parts of the DLGTEMPLATE and WINDOWTEMPLATE statements are described below.

Purpose
This statement marks the beginning of a window template. It defines the name of the window, and its memory and load options.

resourceId *(USHORT)*
is either a unique name or an integer number identifying the resource.

loadoption *(LOADOPTION)*
The default is LOADONCALL.

memoption *(MEMOPTION)*
The default is MOVEABLE.

code page *(USHORT)*
is the code page of the text in the template.

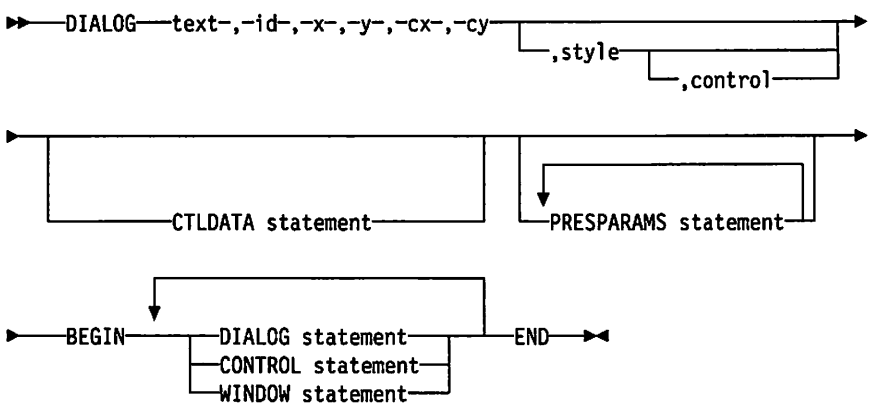
Alternatively, "{" can be used in place of BEGIN and "}" in place of END.

The DLGTEMPLATE and WINDOWTEMPLATE keywords are synonymous.

The DIALOG statement defines a dialog-box window that can be created by an application.

The DIALOG statement has the format:

DIALOG statement

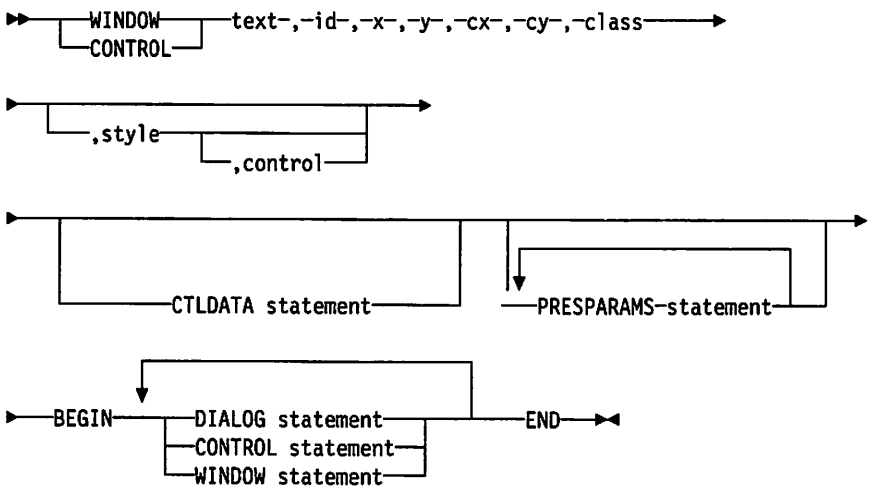


CTLDATA statement
See page 26-21

PRESPARAMS statement
See page 26-21

The WINDOW and CONTROL statements have the format:

WINDOW and CONTROL statements



CTLDATA statement
See page 26-21

PRESPARAMS statement
See page 26-21

Programming Note: The WINDOW and CONTROL keywords are synonymous.

The DIALOG, CONTROL, and WINDOW statements between the BEGIN and END statements are defined as child windows. Presentation parameters always apply to the whole control. They cannot be changed for the individual items within the control.

The parameters of these statements are described below.

Purpose

These statements mark the beginning of a window. They define the starting location on the display screen, its width, its height, and other details such as style.

Note: Not all values may be specified for each statement type. For details, see the call syntax diagrams.

text *(STR)*

is a string, enclosed in double quotes, that is displayed in the title-bar control, if it exists. To insert a double-quote character (") in the text, use two double-quote characters ("").

id *(USHORT)*

item identifier.

x,y *(SHORT)*

are integer numbers specifying the x and y coordinates on the display screen of the lower left corner of the dialog. x and y are in dialog coordinates. The exact meaning of the coordinates depends on the style defined by the style argument. For normal dialogs, the coordinates are relative to the origin of the parent window. For FCF_SCREENALIGN style boxes, the coordinates are relative to the origin of the display screen. With FCF_MOUSEALIGN, the coordinates are relative to the position of the pointer at the time the dialog is created.

cx,cy *(SHORT)*

are integer numbers specifying the width and height of the window.

class *(STR)*

is the class of the window or control to be created.

Programming Note: For a DIALOG statement the class is fixed as WC_FRAME and cannot be specified.

style *(BIT32)*

is any additional window style, frame style, or other class-specific style.

The default style is WS_SYNCPAINT | WS_CLIPSIBLINGS | WS_SAVEBITS | FS_DLGBORDER . If the FS_DLGBORDER or WS_SAVEBITS styles are not required, they should be preceded by the keyword 'NOT'. For example:

- NOT FS_DLGBORDER | FS_BORDER | NOT WS_SAVEBITS

replaces the FS_DLGBORDER default style by the FS_BORDER style and removes the WS_SAVEBITS style. Note that the logic of the 'NOT' keyword is different from the corresponding operator in the C language.

It is not possible to remove the default WS_SYNCPAINT and WS_CLIPSIBLINGS styles.

control *(BIT32)*

Frame Creation Flags (FCF_*; see page 15-2) for the window

This data is placed in the control data field in the correct format for a window of class WC_FRAME.

Programming Note: FCF_SHELLPOSITION has no effect if specified in a template.

Keyboard Resources

RT_KBDSCANXLATE (=17), is defined in BSEDOS.H, and the resource compiler (RC.EXE) recognizes **KBDSCANXLATE**. This type identifies a 256-byte table, that can be used for either primary or secondary scan-code mapping.

The resource id contains three bytes, the least significant byte identifying the type of scan-code mapping table as follows:

0	Primary scan-code mapping
1	Secondary scan-code mapping.

The other two bytes are 0 for the primary mapping table, and the keyboard id (as defined in PMWINP.H) for secondary mapping tables. This is to enable simple support to be provided for future keyboards with conflicting scan codes.

The primary scan-code mapping table in the interrupt handler is stored as a resource of this type. The secondary scan-code mapping table in the interrupt handler is also stored as a resource of this type.

Depending on which keyboard is attached, the resources are loaded when the system is initialized, and transferred to RING 0 byte arrays, where they can be accessed by the interrupt handler as necessary. A default primary scan-code mapping table is transferred if the resource cannot be loaded.

Country Dependent Data.

RT_KBDNAMES (=16), is defined in BSEDOS.H, and the resource compiler (RC.EXE) recognizes **KBDNAMES**.

The keyboard names are stored as a resource of this type and loaded when the system is initialized. The names are transferred to the local storage of PMWIN where they can be accessed as necessary.

Templates, Control Data, and Presentation Parameters

Dialog Template

A dialog template is a data structure used to define a dialog box. Dialog templates can be loaded from resources or created dynamically in memory. Dialog templates define windows of any window class that contain child windows of any class. For standard dialog windows, the dialog window itself is created with the WC_FRAME class, and its children are any of the preregistered control classes.

The dialog template specifies all the information required to create a dialog box and its children.

Dialog Coordinates

Coordinates in a dialog template are specified in *dialog coordinates*. These are based on the default character cell size; a unit in the horizontal direction is ¼ the default character-cell width, and a unit in the vertical direction is ⅛ the default character-cell height. The origin is the bottom left-hand corner of the dialog box.

Dialog Template Format and Contents

A dialog template has these sections:

Header Defines the type of template format and contains information about the location of the other sections of the template. It also contains a summary of the status of the individual controls contained within the dialog box.

Items Defines each of the controls that comprise the dialog box.

Data area Contains the data values associated with each control. Each control defined in the item section contains pointers to the data area section. The data area also contains presentation parameter definitions. The data area is not necessarily a contiguous portion of the template. User data can be placed anywhere in the template if it does not interfere with other defined information.

The sections of a dialog template are illustrated in Figure 26-1 on page 26-19.

Notes:

1. Throughout the dialog template all lengths are in bytes. String lengths do not include any null terminator that may be present. When strings are passed to the Presentation Interface, the length specifications are used and any null terminators are ignored. When strings are returned by the Presentation Interface, length specifications and null terminators are both supplied; therefore, space must be allowed for a null terminator.
2. All offsets are in bytes from the start of the dialog template structure.

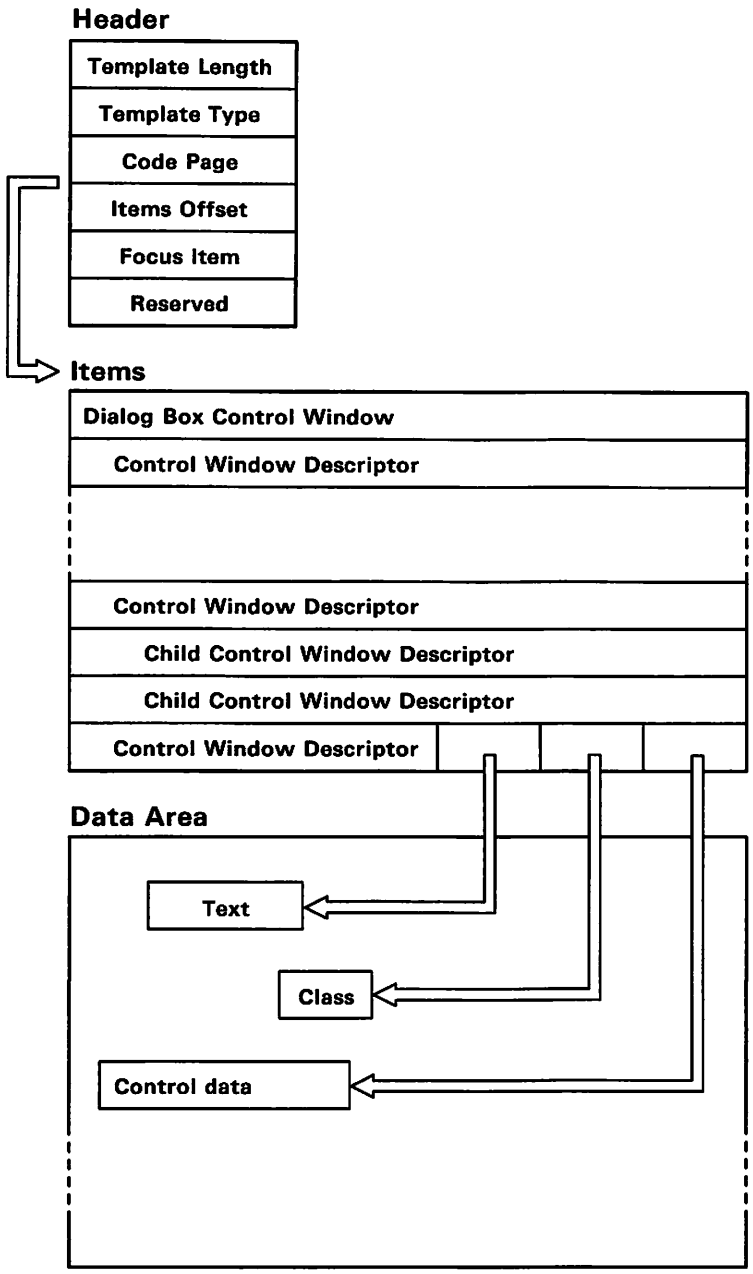


Figure 26-1. Dialog Template

Header

The dialog template header consists of:

Template length *(USHORT)*
The overall length of the dialog template.

Template type *(USHORT)*
The dialog template format type. The format defined is type 0.

Code page *(USHORT)*
The code page of the text in the dialog template.

Items offset *(USHORT)*
The offset of the array of dialog items.

Reserved *(BIT16)*
Must be zero.

Focus item *(USHORT)*
The index in the array of dialog items of the control to receive the focus. If this value is zero, or if the identified control cannot receive the focus, for example because it is a static control, the focus is passed to the first item within the template that can receive the focus.

Reserved *(BIT16)*
Must be zero.

Items

The dialog template items are specified as elements of an array that also defines the hierarchy of the control windows of the dialog box. Each element of the array is a control window descriptor and defines some control or a child of some control, so that every control within the dialog box is described by this array. The first descriptor is the specification of the dialog box itself.

The dialog template items consist of:

Reserved *(BIT16) (16_bit BOOL)*
Must be zero.

Children *(USHORT)*
The number of dialog item child windows that are owned by this dialog item.

This is the number of elements following in the array that are created as child windows of this window. Each window can have any number of child windows, which allows for a tree-structured arrangement.

For example, in Figure 26-1 on page 26-19, assuming that there are no more dialog items than are shown, the first item, the dialog box control window descriptor, has three children. The second item has no children, the third item has two children, and the remaining three items have no children.

Class name length *(USHORT)*
The length of the window class name string.

Class name offset *(USHORT)*
The offset of the window class name string.

Text length *(USHORT)*
The length of the text string.

For controls that allow input of text, this is the current text length, not the maximum text length, and so this value changes when text is put into the control.

Text offset *(USHORT)*
The offset of the text string.

Style *(BIT32) (32_bit BOOL)*
The window style of the control.

16 bits are the standard style bits. The use of the remaining 16 bits depends on the class of the control.

x *(SHORT)*
y *(SHORT)*
The position of the origin of the dialog item. This is specified in dialog coordinates, with x and y relative to the origin of the parent window.

cx *(SHORT)*
cy *(SHORT)*
The size of the dialog item in dialog coordinates; it must be greater than zero.

Identifier *(USHORT)*
An application-defined identifier for the dialog item.

Reserved *(USHORT)*
Must be zero.

Control data offset *(USHORT)*
The offset of the control-specific data for this dialog item. A value of zero indicates that there is no control data for this dialog item.

Data Area

The dialog template data area contains the following different types of object: **text, class name, presentation parameters**, and **control data**. These objects can be placed anywhere within the data area. They do not have to be in contiguous storage, and so an application can place data for its own use between these objects.

The dialog template data area contains:

Text *(STR)*
The textual data associated with a dialog item.

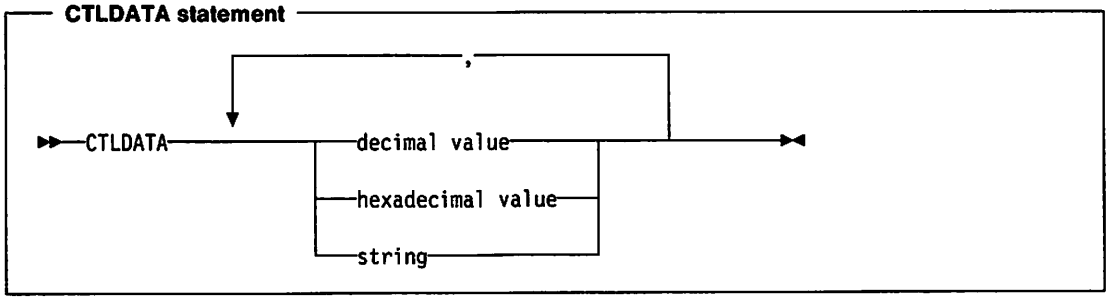
Class name *(STR)*
The name of the window class.

Presentation parameters *(PRESPARAMS)*
Presentation parameters are defined in "Presentation Parameters."

Control data *(CTLDATA)*
For more information, see "Control Data."

Control Data

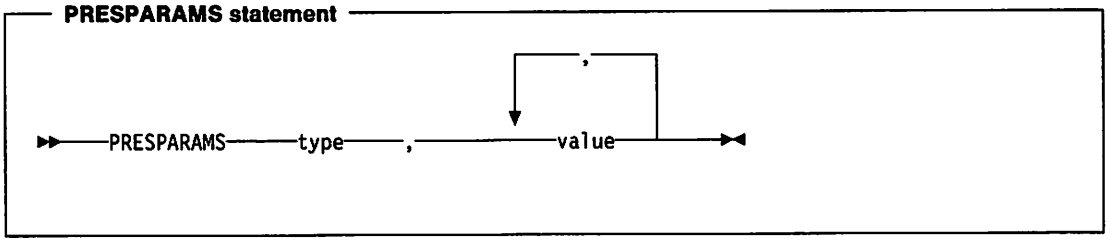
The optional CTLDATA statement is used to define control data for the control. Hexadecimal or decimal word constants follow the CTLDATA statement, separated with commas.



In addition to hexadecimal or decimal data, the CTLDATA statement can be followed by the MENU keyword, followed by a menu template in a BEGIN/END block. This creates a menu template as the window's control data.

Presentation Parameters

The optional PRESPARAMS statement is used to define presentation parameters. The syntax of the PRESPARAMS statement is as follows.



A presentation parameter consists of:

type *(ULONG)*
The presentation parameter attribute type. See the **param** data type for a description of valid types.

A string can be used to specify the type for a user type. If this is done, the string type is converted into a string atom when the dialog template is read into memory. Thereafter this presentation parameter is referred to by this string atom. The application can use the atom manager API to match the string and the string atom.

value *(LONG or STRL)*
One or more values depending upon the attribute type.

If the value is enclosed in quotes it is a zero-terminated string. Otherwise, it is converted to a LONG. There may be more than one value, depending upon the type. See **param** data type for a description of the values required for system-defined presentation parameters.

Examples: The following are examples of PRESPARAMS statements:

```
PRESPARAMS PP_BORDERCOLOR,  0x00ff00ffL
PRESPARAMS PP_FONTNAMESIZE, "12.Helv"
PRESPARAMS "my color",      0x00ff00ffL
PRESPARAMS "my param",      0, 1, 2, 3, "Hi there"
```

Parent/Child/Owner Relationship

The format of the DLGTEMPLATE and WINDOWTEMPLATE resources is very general to allow tree-structured relationships within the resource format. The general layout of the templates is:

```
WINDOWTEMPLATE id
BEGIN
    WINDOW winTop             the top-level window
    BEGIN
        WINDOW wind1
        WINDOW wind2
        WINDOW wind3
        BEGIN
            WINDOW wind4
        END
        WINDOW wind5
    END
END
```

In this example, the top-level window is identified by **winTop**. It has four child windows: **wind1**, **wind2**, **wind3**, and **wind5**. **wind3** has one child window, **wind4**. When each of these windows is created, the parent and the owner are set to be the same.

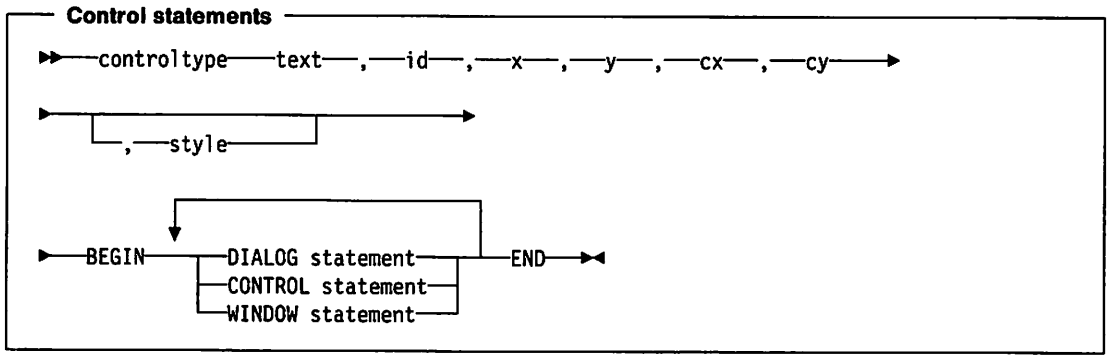
The only time when the parent and owner windows are not the same is when frame controls are automatically created by a frame window.

Note that the WINDOW statements in the example above could also have been CONTROL or DIALOG statements.

Predefined Control Statements

In addition to the general form of the CONTROL statement, there are special control statements for commonly used controls. These statements define the attributes of the child control windows that appear in the window.

Control statements have this general form:



The LISTBOX control statement is an exception to this general form because it does not have a text field.

controltype
is one of the keywords described below, defining the type of the control.

text *(STR)*
is a string specifying the text to be displayed. The string must be enclosed in double quotation marks. The manner in which the text is displayed depends on the particular control, as detailed below.

To indicate the mnemonic for each item, insert the tilde character (~) in the string preceding the mnemonic character.

id *(USHORT)*
is a unique integer number identifying the control.

x,y *(SHORT)*
are integer numbers specifying the x and y coordinates of the lower left corner of the control, in dialog coordinates. The coordinates are relative to the origin of the dialog.

cx,cy *(SHORT)*
are integer numbers specifying the width and height of the control.

The x, y, cx, and cy fields can use addition and subtraction operators (+ and −). For example, 15 + 6 can be used for the x field.

Styles can be combined using the "|" operator.

The control type keywords are shown below, with their classes and default styles:

FRAME

Class	WC_FRAME
Default style	WS_VISIBLE

LTEXT

Class	WC_STATIC
Default style	SS_TEXT, DT_LEFT, WS_GROUP, WS_VISIBLE

RTEXT

Class	WC_STATIC
Default style	SS_TEXT, DT_RIGHT, WS_GROUP, WS_VISIBLE

CTEXT

Class	WC_STATIC
Default style	SS_TEXT, DT_CENTER, WS_GROUP, WS_VISIBLE

CHECKBOX

Class	WC_BUTTON
Default style	BS_CHECKBOX, WS_TABSTOP, WS_VISIBLE

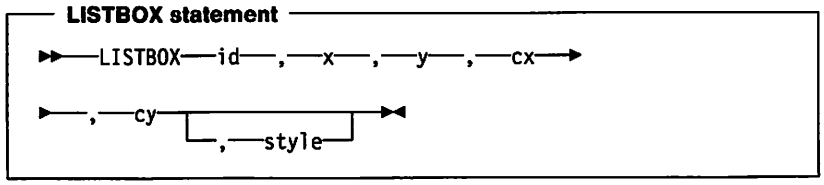
PUSHBUTTON

Class	WC_BUTTON
Default style	BS_PUSHBUTTON, WS_TABSTOP, WS_VISIBLE

LISTBOX

Format

The LISTBOX control statement does not contain a text field, so its form is:

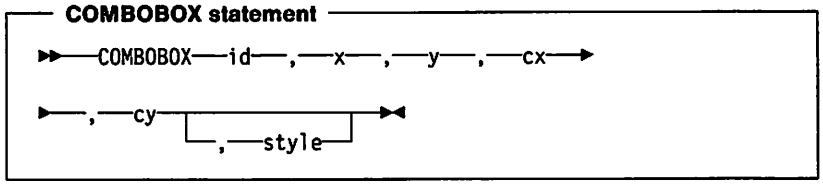


The fields have the same meaning as in the other control statements.

Class WC_LISTBOX
Default style LBS_NOTIFY, LBS_SORT, WS_VSCROLL, WS_BORDER, WS_VISIBLE

COMBOBOX

Format The COMBOBOX control statement has the following form:



The fields have the same meaning as in the other control statements.

Class WC_COMBOBOX
Default style CBS_SIMPLE, WS_TABSTOP, WS_VISIBLE

GROUPBOX

Class WC_STATIC
Default style SS_GROUPBOX, WS_TABSTOP, WS_VISIBLE

DEFPUSHBUTTON

Class WC_BUTTON
Default style BS_DEFAULT, BS_PUSHBUTTON, WS_TABSTOP, WS_VISIBLE

RADIOBUTTON

Class WC_BUTTON
Default style BS_RADIOBUTTON, WS_TABSTOP, WS_VISIBLE

AUTORADIOBUTTON

Class WC_BUTTON
Default style BS_AUTORADIOBUTTON, WS_TABSTOP, WS_VISIBLE

ENTRYFIELD

Class WC_ENTRYFIELD
Default style WS_TABSTOP, ES_LEFT, WS_VISIBLE

ICON

Class WC_STATIC
Default style SS_ICON, WS_VISIBLE

Examples: The following is a complete example of a DIALOG statement:

```
DLGTEMPLATE errmess
BEGIN
    DIALOG  "Disk Error", 100, 10, 10, 300, 110
    BEGIN
        CTEXT "Select One:", 1, 10, 80, 280, 12
        RADIOBUTTON "Retry", 2, 75, 50, 60, 12
        RADIOBUTTON "Abort", 3, 75, 30, 60, 12
        RADIOBUTTON "Ignore", 4, 75, 10, 60, 12
    END
END
```

This is an example of a WINDOWTEMPLATE statement that is used to define a specific kind of window frame. Calling Load Dialog with this resource automatically creates the frame window, the frame controls, and the client window (of class MyClientClass).

```
WINDOWTEMPLATE wind1
BEGIN
    FRAME "My Window", 1, 10, 10, 320, 130
    CTLDATA FCF_STANDARD | FCF_VERTSCROLL
    BEGIN
        WINDOW "", FID_CLIENT, 0, 0, 0, 0, "MyClientClass", style
    END
END
```

This example creates a resource template for a parallel dialog identified by the constant **parallel1**. It includes a frame with a title bar, a system menu, and a dialog-style border. The parallel dialog has three autoradiobuttons in it.

```
DLGTEMPLATE parallel1
BEGIN
    DIALOG "Parallel Dialog", 1, 50, 50, 180, 110
    CTLDATA FCF_TITLEBAR | FCF_SYSMENU | FCF_DLGBORDER
    BEGIN
        AUTORADIOBUTTON "Retry", 2, 75, 80, 60, 12
        AUTORADIOBUTTON "Abort", 3, 75, 50, 60, 12
        AUTORADIOBUTTON "Ignore", 4, 75, 30, 60, 12
    END
END
```

Resource (.RES) File Specification

The format for the .RES file is:

```
(/TYPE NAME FLAGS SIZE BYTES/)+
```

Where:

TYPE is either a null-terminated string or an ordinal, in which instance the first byte is X'FF' followed by an INT that is the ordinal.

```
/* Predefined resource types */
#define RT_POINTER      MAKEINTRESOURCE( 1 )
#define RT_BITMAP       MAKEINTRESOURCE( 2 )
#define RT_ICON         MAKEINTRESOURCE( 3 )
#define RT_MENU         MAKEINTRESOURCE( 4 )
#define RT_DIALOG       MAKEINTRESOURCE( 5 )
#define RT_STRING       MAKEINTRESOURCE( 6 )
#define RT_FONTDIR      MAKEINTRESOURCE( 7 )
#define RT_FONT         MAKEINTRESOURCE( 8 )
#define RT_ACCELTABLE   MAKEINTRESOURCE( 9 )
#define RT_DLGINCLUDE   MAKEINTRESOURCE(10 )
#define RT_KBDNAMES     MAKEINTRESOURCE(16 )
#define RT_KBDSCANXLATE MAKEINTRESOURCE(17 )
```

NAME is the same format as TYPE. There are no predefined ordinals.

FLAGS is an unsigned value containing the memory manager flags:

```
#define NSTYPE      X'0007'   /* Segment type mask           */
#define NSCODE      X'0000'   /* Code segment                */
#define NSDATA      X'0001'   /* Data segment                */
#define NSITER      X'0008'   /* Iterated segment flag       */
#define NSMOVE      X'0010'   /* Moveable segment flag       */
#define NSPURE      X'0020'   /* Pure segment flag           */
#define NSPRELOAD   X'0040'   /* Preload segment flag        */
#define NSEXRD      X'0080'   /* Execute-only (code segment), */
                              /* or read-only (data segment) */
#define NSRELOC     X'0100'   /* Segment has relocations     */
#define NSDEBUG     X'0200'   /* Segment has debug info      */
#define NSDPL       X'0C00'   /* 286 DPL bits                */
#define NSDISCARD   X'1000'   /* Discard bit for segment     */
```

SIZE is a LONG value defining how many bytes follow in the resource.

BYTES is the stream of bytes that makes up the resource.

Any number of resources can appear one after another in the .RES file.

Chapter 27. Graphics Orders

Graphics Orders

Introduction

This chapter describes the format of the graphics orders.

Graphics orders are used in the following circumstances:

- Using GpiGetData or GpiPutData calls for bulk transfer of part or all of graphics segment data (unless this is simply being copied without being changed).
- Editing segments with GpiQueryElement and GpiElement.
- Generating metafiles (other than through the Presentation Manager API), or examining their contents. The data part of Graphics Data structured fields within the metafile (see "Metafile Data Format" on page D-2) consists of graphics orders.

When primitive or attribute functions (plus certain other functions) are specified at the programming interface, and the drawing mode (see GpiSetDrawingMode) is set to **drawandretain**, graphics orders are constructed and placed in the current graphics segment. One API call often causes a single order to be generated. Sometimes, however, several orders are necessary: an example of this is where a GpiPolyLine call is issued, which specifies more strokes than there is room for, in a single order.

In either case, the order or orders generated by a single API call comprise a single **element**, unless the application specifically starts an element using the GpiBeginElement call. In this case the element consists of all of the orders generated between this and the following GpiEndElement call. A GpiQueryElement call returns the orders that comprise an element; the application may edit these, and return them to the segment with GpiElement. The Begin Element – End Element orders that surround a multi-order element in the segment are never passed between the application and the system on GpiQueryElement and GpiElement calls.

No double word or word alignment can be assumed for orders either within segments or during editing.

Data Types

All data types Intel format unless noted otherwise.

GBIT1	1-bit field.
GBIT16	16-bit field.
GBIT2	2-bit field.
GBIT32	32-bit field.
GBIT4	4-bit field.
GBIT5	5-bit field.
GBIT6	6-bit field.
GBIT7	7-bit field.
GBIT8	8-bit field.
GCHAR	Signed 1-byte integer value.

GDELPOINT Offset point structure.

dx *(GCHAR)*
x coordinate offset.

dy *(GCHAR)*
y coordinate offset.

GFIXED Signed integer fraction (16:16). (This can be treated as a *GLONG* where the value has been multiplied by 65 536.)

GHBITMAP Bit-map handle, which is the same as *GULONG*.

GINDATT Individual attribute value. For attribute types color and background color, this is the same as *GINDEX3*. For attribute types mix and background color, this is the same as *GUCHAR*.

GINDEX3 Unsigned 3-byte integer value.

GLENGTH1 1-byte length.

GLENGTH2 2-byte length, in S/370 format; that is, the high-order byte precedes the low-order byte in storage.

GLONG Signed 4-byte integer value.

GPOINT Point structure.

x *(GROSOL)*
x coordinate.

y *(GROSOL)*
y coordinate.

GPOINTB Point in bit-map structure.

x *(GLONG)*
x coordinate.

y *(GLONG)*
y coordinate.

GREAL Real (single precision floating point).

This data type is in Intel format.

GROF Number representation, which is the same as the *GFIXED* data type.

GROFUFS Number representation, which is either *GFIXED*, *GUFIXEDS* or *GREAL* data type, depending on the presentation-space format.

GROUFS Number representation, which is either the *GUFIXEDS* or *GREAL* data type, depending on the presentation-space format.

GROSOL Number representation, which is either the *GSHORT* or the *GLONG* data type, depending on the presentation-space format; see PS_FORMAT in the **Options** parameter of the GpiCreatePS call.

GSHORT Signed 2-byte integer value.

GSTR String with an explicit length count.

GUCHAR Unsigned 1-byte integer value.

GULONG Unsigned 4-byte integer value.

GUFIXEDS Unsigned integer fraction (8:8) which can be treated as a *GUSHORT* data type, where the value has been multiplied by 256.

GUSHORT370 Unsigned 2-byte integer value, in S/370 format; that is, the high-order byte precedes the low-order byte in storage.

Arc at a Given Position / Arc at Current Position

These orders construct an arc starting at a given position or at the current position.

Arc at a Given Position (GARC)
X'C6'(len, p0, p1, p2)

Arc at Current Position (GCARC)
X'86'(len, p1, p2)

Parameters

len *(GLENGTH1)*
Length of following data.

p0 *(GPOINT)*
Coordinates of start point.

This parameter is only present in a Arc at a Given Position order.

p1 *(GPOINT)*
Coordinates of intermediate point.

p2 *(GPOINT)*
Coordinates of end point.

Begin Area

This order indicates the start of a set of primitives that define an area boundary.

Begin Area (GBAR)
X'68'(flags)

Parameters

flags
Internal flags.

res1 *(GBIT1)*
Reserved for compatibility:

1 Only valid value.

boundary *(GBIT1)*
Boundary-line draw indicator:

0 Do not draw boundary lines
1 Draw boundary lines.

inside *(GBIT1)*
Mode shading:

0 Alternate mode
1 Winding mode.

res2 *(GBIT5)*
Reserved.

00000 Reserved value.

Begin Element

This order indicates the beginning of a set of primitives that define an element.

Begin Element (GBEL)
X'D2'(len, type, descr)

Parameters

len *(GLENGTH1)*
Length of following data.

type *(GLONG)*
Element type.

Values are:

X'0000FD01'	Line bundle
X'0000FD02'	Character bundle
X'0000FD03'	Marker bundle
X'0000FD04'	Area bundle
X'0000FD05'	Image bundle
X'00000007'	Call segment
X'00000081'	Polyline
X'00000085'	Polyfillet
X'000000A4'	Polyfillet sharp
X'000000A5'	Polyspline
X'00000082'	Polymarker
X'00000087'	Full arc
X'00000091'	Image
X'000000B1'	Character string at current position
X'000000F1'	Character string at given position
X'81xxxxxx' – X'FFxxxxxx'	Indicates user defined elements
other	Reserved values.

descr *(GSTR)*
Element description data.

This is optional.

Begin Image at Given Position / Begin Image at Current Position

These orders identify the start of an image definition at a given position or at the current position.

Begin Image at Given Position (GBIMG)
X'D1'(len, p0, format, res, width, height)

Begin Image at Current Position (GCBIMG)
X'91'(len, format, res, width, height)

Parameters

len *(GLENGTH1)*
Length of following data.

p0 *(GPOINT)*
Point at which the image is to be placed.

This parameter is only present in a Begin Image at Given Position order.

format *(GUCHAR)*
Format of the image data.

X'00' One bit in the data represents one pel on the output medium. It is the only valid value.

res *(GBIT8)*
Reserved.

X'00' Only valid value.

width *(GUSHORT370)*
Width of the image data.

This is the width in pels.

height *(GUSHORT370)*
Height of the image data.

This is the height in pels.

Begin Path

This order sets the drawing process into path state.

Begin Path (GBPTH)
X'D0'(len, res, pthid)

Parameters

len *(GLENGTH1)*
Length of following data.

res *(GBIT16)*
Reserved.

X'0000' Only valid value.

pthid *(GLONG)*
Path identifier.

1 Only valid value.

Bézier Curve at Given Position / Bézier Curve at Current Position

These orders generate a curve that starts at a given position or at the current position.

Bézier Curve at Given Position (GBEZ)
X'E5'(len, p0, p1, p2, p3, p4, p5, p6, pn-2, pn-1, pn)

Bézier Curve at Current Position (GCBEZ)
X'A5'(len, p1, p2, p3, p4, p5, p6, pn-2, pn-1, pn)

Parameters

len *(GLENGTH1)*
Length of following data.

p0 *(GPOINT)*
Coordinate data of first curve start.

This parameter is only present in a Bézier Curve at Given Position.

p1 *(GPOINT)*
Coordinate data of first control point.

p2 *(GPOINT)*
Coordinate data of second control point.

p3 *(GPOINT)*
Coordinate data of first curve end.

p4, p5, p6 *(GPOINT)*
Coordinate data for second curve.

pn-2, pn-1, pn *(GPOINT)*
Coordinate data for last curve.

Remarks

Further points are used in groups of three to form a polycurve. Each group of points, together with the last point of the previous curve, generates a new curve.

Bitblt

This order copies a rectangle of a bit map into DOCS.

Bitblt (GBBLT)
X'D6'(len, flags, mix, bmid, trans, p1, p2, source1, source2)

Parameters

len *(GLENGTH1)*
Length of following data.

flags *(GBIT16)*
Reserved.

X'0000' Only valid value.

mix *(GBIT16)*
Mix mode.

Values are:

X'00CC'	Source
X'00C0'	Source and pattern
X'00CA'	Source where pattern1
X'000C'	Source where pattern0
X'00E2'	Pattern where source1
X'00B8'	Pattern where source0
other	Reserved values.

bmId *(GHBITMAP)*
Bit-map identifier.

trans *(GBIT32)*
Transfer mode.

Values are:

X'00000000'	OR
X'01000000'	AND
X'02000000'	Ignore
other	Reserved values.

p1 *(GPOINT)*
Target rectangle bottom left corner.

p2 *(GPOINT)*
Target rectangle top right corner.

source1 *(GPOINTB)*
Source rectangle bottom left corner.

source2 *(GPOINTB)*
Source rectangle top right corner.

Box at Given Position / Box at Current Position

These orders define a box with square or round corners, drawn with its first corner at a given position or at the current position.

Box at Given Position (GBOX)
X'C0'(len, control, res, p0, p1, haxis, vaxis)

Box at Current Position (GCBOX)
X'80'(len, control, res, p1, haxis, vaxis)

Parameters

len *(GLENGTH1)*
Length of following data.

control
Internal flags.

res1 *(GBIT1)*
Reserved.

0 Only valid value.

fill *(GBIT1)*
Values:

0 No fill
1 Fill.

boundary *(GBIT1)*
Values:

0 No boundary
1 Boundary.

res2 *(GBIT5)*
Reserved.

00000 Only valid value.

res *(GBIT8)*
Reserved.

X'00' Only valid value.

p0 *(GPOINT)*
Coordinate data of box origin.

This parameter is only present in a Box at Given Position order.

p1 *(GPOINT)*
Coordinate data of box corner.

haxis *(GROSOL)*
Length of horizontal axis of ellipse.

vaxis *(GROSOL)*
Length of vertical axis of ellipse.

Call Segment

This order calls one segment from another.

Call Segment (GCALLS)
X'07'(len, res, segname)

Parameters

len *(GLENGTH1)*
Length of following data.

res *(GBIT16)*
Reserved value.

X'0000' Only valid value.

segname *(GLONG)*
Name of segment that is to be called.

The name cannot be zero.

Character String at Given Position / Character String at Current Position

These orders draw a character string at a given position or at the current position.

Character String at Given Position (GCHST)
X'C3'(len, p0, cp)

Character String at Current Position (GCCHST)
X'83'(len, cp)

Parameters

len *(GLENGTH1)*
Length of following data.

p0 *(GPOINT)*
Point at which character string is to be placed.

This parameter is only present in a Character String at Given Position order.

cp *(GSTR)*
Code points of each character in the string.

Character String Extended at Given Position / Character String Extended at Current Position

These orders define a character string to be drawn at a given position or at the current position.

Character String Extended at Given Position (GCHSTE)
X'FEF0'(len1, p0, flags, res, p1, p2, len2, cp, pad, vect)

Character String Extended at Current Position (GCCHSTE)
X'FEB0'(len1, flags, res, p1, p2, len2, cp, pad, vect)

Parameters

len1 *(GLENGTH2)*
Length of following data.

p0 *(GPOINT)*
Point at which the character string is to be placed.

This parameter is only present in a Character String Extended at Given Position order.

flags
Extra functions:

rect *(GBIT1)*
Values:

- **0** Do not draw background rectangle
- **1** Draw background rectangle.

clip *(GBIT1)*
Values:

0 Do not clip to rectangle
1 Clip to rectangle.

res1 *(GBIT1)*
Reserved.

0 Only valid value.

lvcp *(GBIT1)*
Values:

0 Move current position
1 Leave current position.

res2 *(GBIT4)*
Reserved.

0000 Only valid value.

res *(GBIT8)*
Reserved.

X'00' Only valid value.

p1 *(GPOINT)*
Coordinate data of rectangle corner.

p2 *(GPOINT)*
Coordinate data of rectangle corner.

len2 *(GLENGTH2)*
Length of code-point data.

cp *(GSTR)*
Code-point data.

pad *(GBIT8)*
Pad byte.

Only needs to be included if **cp** is an odd number of bytes.

vect *(GROSOL*n)*
Vector of character increments.

n is the number of code points present in the **cp** parameter.

Character String Move at Given Position / Character String Move at Current Position

These orders draw a character string starting from a given point or from the current point and moves the current position to the end of the string.

Character String Move at Given Position (GCHSTM)
X'F1'(len, p0, cp)

Character String Move at Current Position (GCCHSTM)
X'B1'(len, cp)

Parameters

len *(GLENGTH1)*
Length of following data.

p0 *(GPOINT)*
Point at which the character string is to be placed.

This parameter is only present in a Character String Move at Given Position order.

cp *(GSTR)*
Code points of each character in the string.

Close Figure

This order delimits the end of a closed figure.

Close Figure (GCLFIG)
X'7D'(res)

Parameters

res *(GBIT8)*
Reserved.

X'00' Only valid value.

Comment

This order enables data to be stored within a segment.

Comment (GCOMT)
X'01'(len, data)

Parameters

len *(GLENGTH1)*
Length of following data.

data *(GBIT8*len)*
Comment data.

Remarks

This order is treated as a no-operation.

End Area

This order indicates the end of a set of primitives that define an area boundary.

End Area (GEAR)
X'60'(len, data)

Parameters

len *(GLENGTH1)*
Length of following data. It is normally zero.

data *(GBIT8*len)*
Reserved.

X'00' Only valid value.

End Element

This order identifies the end of a set of primitives that define an element.

End Element (GEEL)
X'49'(res)

Parameters

res *(GBIT8)*
Reserved.

X'00' Only valid value.

End Image

This order identifies the end of an image definition.

End Image (GEIMG)
X'93'(len, data)

Parameters

len *(GLENGTH1)*
Length of following data. It is normally zero.

data *(GBIT8*len)*
Reserved.

X'00' Only valid value.

End of Symbol Definition

This order indicates the end of a set of orders defining a graphic symbol.

End of Symbol Definition (GESD)
X'FF'

Remarks

This order is only valid in the context of symbol definitions.

End Path

This order ends the definition of a path.

End Path (GEPTH)
X'7F'(res)

Parameters

res *(GBIT8)*
Reserved.

X'00' Only valid value.

End Prolog

This order indicates the end of the prolog of a segment.

End Prolog (GEPROL)
X'3E'(res)

Parameters

res *(GBIT8)*
Reserved.

X'00' Only valid value.

Escape

This order provides facilities for registered and unregistered escape functions.

Escape (GESCP)
X'D5'(len, type, rid, parms)

Parameters

len *(GLENGTH1)*
Length of following data.

type *(GUCHAR)*
Type identifier:

80 Registered value
Other All other values are unregistered.

rid *(GUCHAR)*
Registered identifier:

01 Set pel
02 BITBLT function.

parms *(GSTR)*
Parameters of escape.

Extended Escape

This order provides facilities for registered and unregistered escape functions.

Extended Escape (GEESCP)
X'FED5'(len, type, rid, parms)

Parameters

len *(GLENGTH2)*
Length of following data.

type *(GUCHAR)*
Type identifier:

80 Registered value
Other All other values are unregistered.

rid *(GUCHAR)*
Registered identifier.

No registered extended escapes are used by OS/2 Version 1.2.

parms *(GSTR)*
Parameters of escape.

Fill Path

This order fills the interior of the specified path.

Fill Path (GFPTH)
X'D7'(len, flags, res, pthid)

Parameters

len *(GLENGTH1)*
Length of following data.

flags
Extra functions:

res1 *(GBIT1)*
Reserved.

0 Only valid value.

inside *(GBIT1)*
Values:

0 Alternate mode
1 Winding mode.

mod *(GBIT1)*
Values:

0 Do not modify before filling
1 Modify path before filling.

res2 *(GBIT5)*
Reserved.

00000 Only valid value.

res *(GBIT8)*
Reserved.

X'00' Only valid value.

pthid *(GLONG)*
Path identifier.

1 Only valid value.

Fillet at Given Position / Fillet at Current Position

These orders draw a curved line tangential to a specified set of straight lines, at the given position or at the current position.

Fillet at Given Position (GFLT)
X'C5'(len, p0, p1, pn)

Fillet at Current Position (GCFLT)
X'85'(len, p1, pn)

Parameters

len *(GLENGTH1)*
Length of following data.

p0 *(GPOINT)*
Coordinate data of line start.

This parameter is only present in a Fillet at Given Position order.

p1 *(GPOINT)*
Coordinate data of first line end.

pn *(GPOINT)*
Coordinate data of final line end.

Full Arc at Given Position / Full Arc at Current Position

These orders construct a full circle or an ellipse, with the center at a specified point or at the current position.

Full Arc at Given Position (GFARC)
X'C7'(len, p0, m)

Full Arc at Current Position (GCFARC)
X'87'(len, m)

Parameters

len *(GLENGTH1)*
Length of following data.

p0 *(GPOINT)*
Coordinate data of the center of the circle/ellipse.

This parameter is only present in a Full Arc at Given Position order.

m *(GROFUFS)*
Multiplier.

Image Data

This order provides bit data for an image.

Image Data (GIMD)
X'92'(len, data)

Parameters

len *(GLENGTH1)*
Length of following data.

data *(GBIT8*len)*
Image data.

Label

This order is used to label an element within a segment.

Label (GLBL)
X'D3'(len, ldata)

Parameters

len *(GLENGTH1)*
Length of following data.

ldata *(GLONG)*
Label value.

Line at Given Position / Line at Current Position

These orders define one or more connected straight lines, drawn from the given position or from the current position.

Line at Given Position (GLINE)
X'C1'(len, p0, p1, pn)

Line at Current Position (GCLINE)
X'81'(len, p1, pn)

Parameters

len *(GLENGTH1)*
Length of following data.

p0 *(GPOINT)*
Coordinate data of line start.

This parameter is only present in a Line at Given Position order.

p1 *(GPOINT)*
Coordinate data of first line end.

pn *(GPOINT)*
Coordinate data of final line end.

Marker at Given Position / Marker at Current Position

These orders draw the current marker symbol at one or more positions starting from the given position or from the current position.

Marker at Given Position (GMRK)
X'C2'(len, p0, p1, pn)

Marker at Current Position (GCMRK)
X'82'(len, p1, pn)

Parameters

len *(GLENGTH1)*
Length of following data.

p0 *(GPOINT)*
Coordinate data of first marker.

p1 *(GPOINT)*
Coordinate data of second marker.

pn *(GPOINT)*
Coordinate data of final marker.

Modify Path

This order modifies the path according to the value of the mode.

Modify Path (GMPTH)
X'D8'(len, mode, res, pthId)

Parameters

len *(GLENGTH1)*
Length of following data.

mode *(GUCHAR)*
Mode of path modification:

X'06' Stroke the path
Other All other values are reserved.

res *(GBIT8)*
Reserved.

X'00' Only valid value.

pthId *(GLONG)*
Path identifier.

1 Only valid value.

No-Operation

This order is a no-operation.

No-Operation (GNOP1)
X'00'

Outline Path

This order draws the outline of the specified path.

Outline Path (GOPTH)
X'D4'(len, flags, res, pthid)

Parameters

len *(GLENGTH1)*
Length of following data.

flags *(GBIT8)*
Function flags:

X'00' Only valid value.

res *(GBIT8)*
Reserved.

X'00' Only valid value.

pthid *(GLONG)*
Path identifier.

1 Only valid value.

Partial Arc at Given Position / Partial Arc at Current Position

These orders draw a line from the Given Position or the Current Position to the start of an arc, and then draw the arc.

Partial Arc at Given Position (GPARC)
X'E3'(len, p0, p1, m, start, sweep)

Partial Arc at Current Position (GCPARC)
X'A3'(len, p1, m, start, sweep)

Parameters

len *(GLENGTH1)*
Length of following data.

p0 *(GPOINT)*
Coordinate data of start of line.

This parameter is only present in a Partial Arc at Given Position order.

p1 *(GPOINT)*
Coordinate data of center of arc.

m *(GROFUFS)*
Multiplier.

start *(GROF)*
Start angle.

sweep *(GROF)*
Sweep angle.

Pop

This order enables data to be popped from the Segment Call Stack.

Pop (GPOP)
X'3F'(res)

Parameters

res *(GBIT8)*
Reserved.

X'00' Only valid value.

Remarks

The data is placed into an attribute or Drawing Process Control.

Relative Line at Given Position / Relative Line at Current Position

These orders define one or more connected straight lines, at the given position or at the current position.

Relative Line at Given Position (GRLINE)
X'E1'(len, p0, off0, off1, offn)

Relative Line at Current Position (GCRLINE)
X'A1'(len, off0, off1, offn)

Parameters

len *(GLENGTH1)*
Length of following data.

p0 *(GPOINT)*
Coordinate data of line start.

This parameter is only present in a Relative Line at Given Position order.

off0 *(GDELPOINT)*
Offset data.

This offset is to the first line end, relative to its start point.

off1 *(GDELPOINT)*
Offset data.

This offset is to the second line end, relative to the first line end.

offn *(GDELPOINT)*
Offset data.

This offset is to the 'n'th line end, relative to the 'n-1'th line end.

Remarks

The end point of each line is given as an offset from the start of the line, rather than as absolute coordinates.

Set Arc Parameter / Push and Set Arc Parameter

These orders set, or push and set, the values of the current arc parameters.

Set Arc Parameter (GSAP)
X'22'(len, p, q, r, s)

Push and Set Arc Parameter (GPSAP)
X'62'(len, p, q, r, s)

Parameters

len *(GLENGTH1)*
Length of following data.

p *(GROSOL)*
P-parameter of arc transform.

q *(GROSOL)*
Q-parameter of arc transform.

r *(GROSOL)*
R-parameter of arc transform.

s *(GROSOL)*
S-parameter of arc transform.

Remarks

The values of the current arc parameters are pushed on to the Segment Call stack by the Push and Set order only. Both orders then set the current arc parameters to the values specified in the order.

The value of these parameters determines the shape of subsequent orders drawn using Arc at a Given Position / Arc at Current Position or Full Arc at Given Position / Full Arc at Current Position or Partial Arc at Given Position / Partial Arc at Current Position.

Set Background Color / Push and Set Background Color

These orders set, or push and set, the value of the current background color attribute.

Set Background Color (GSBCOL)
X'25'(len, color)

Push and Set Background Color (GPSBCOL)
X'65'(len, color)

Parameters

len *(GLENGTH1)*
Length of following data.

2 Only valid value.

color *(GUSHORT)*
Color-table index:

Except for the special values, the values X'0000' through X'nnnn' are allowed keys; that is, as many values as are allowed by the size of the LCT.

Special Values

X'FF00'	Drawing default
X'FF0x'	As value X'000x', where x is in the range 1 through 7
X'FF08'	Reset color, as value X'0000'
Other	All other values are reserved.

Set Background Indexed Color / Push and Set Background Indexed Color

These orders set, or push and set, the value of the current background color attribute.

Set Background Indexed Color (GSBICOL)
X'A7'(len, flags, index)

Push and Set Background Indexed Color (GPSBICOL)
X'E7'(len, flags, index)

Parameters

len *(GLENGTH1)*
Length of following data.

flags
Values:

default *(GBIT1)*
Options:

0 Use specified **index**
1 Use drawing default color.

spec *(GBIT1)*
Options:

0 Use index directly
1 Special value.

res *(GBIT6)*
Reserved.

000000 Only valid value.

index *(GINDEX3)*
Value for color index.

The value is a direct index into the current color table or a special value.

The special values are:

1 Black
2 White
4 All ones
5 All zeros.

Remarks

The value of the current background color attribute is pushed on to the stack by the Push and Set order only. Both orders then set the current background color attribute to the value specified in the order.

Set Background Mix / Push and Set Background Mix

These orders set, or push and set, the value of the current background mix attribute.

Set Background Mix (GSBMX)
X'0D'(mode)

Push and Set Background Mix (GPSBMX)
X'4D'(mode)

Parameters

mode *(GUCHAR)*
Mix-mode value:

X'00' Drawing default
X'01' OR
X'02' Overpaint
X'03' Reserved
X'04' Exclusive-OR
X'05' Leave Alone

X'06' AND
X'07' Subtract
X'08' Source AND (inverse destination)
X'09' All zeros
X'0A' Inverse (source OR destination)
X'0B' Inverse (source XOR destination)
X'0C' Inverse destination
X'0D' Source OR (inverse destination)
X'0E' Inverse source
X'0F' (Inverse source) OR destination
X'10' Inverse (source AND destination)
X'11' All ones.
All other values are reserved.

Remarks

The value of the current background mix attribute is pushed on to the Segment Call stack by the Push and Set order only. Both orders then set the current background mix attribute to the value specified in the order.

Set Character Angle / Push and Set Character Angle

These orders set, or push and set, the value of the current character angle attribute.

Set Character Angle (GSCA)
X'34'(len, ax, ay)

Push and Set Character Angle (GPSCA)
X'74'(len, ax, ay)

Parameters

len *(GLENGTH1)*
Length of following data.

ax *(GROSOL)*
x coordinate of point.

This point defines the angle of the character string.

ay *(GROSOL)*
y coordinate of point.

This point defines the angle of the character string.

Remarks

The value of the current character angle attribute is pushed on to the Segment Call Stack by the Push and Set order only. Both orders then set the value of the current character angle to the value specified in the order.

Set Character Cell / Push and Set Character Cell

These orders set, or push and set, the value of the current character cell-size attribute.

Set Character Cell (GSCC)
X'33'(len, cellx, celly, cellxf, cellyf, flags, res)

Push and Set Character Cell (GPSCC)
X'03'(len, cellx, celly, cellxf, cellyf, flags, res)

Parameters

len *(GLENGTH1)*
Length of following data.

cellx *(GROSOL)*
X part of character cell-size attribute.

celly *(GROSOL)*
Y part of character cell-size attribute.

cellxf *(GUSHORT)*
Fractional X part of character cell-size attribute.

This parameter is optional.

cellyf *(GUSHORT)*
Fractional Y part of character cell-size attribute.

This parameter must be present if **cellxf** parameter is present.

flags
Internal flags.

This parameter is optional.

setzero *(GBIT1)*
Values:

0 A cell size of zero sets drawing default
1 A cell size of zero sets to zero.

res *(GBIT7)*
Reserved.

0000000 Only valid value.

res *(GBIT8)*
Reserved value.

This parameter must be present if **flags** parameter is present.

X'00' Only valid value.

Remarks

The value of the current character cell-size attribute is pushed on to the Segment Call Stack by the Push and Set order only. Both orders then set the value of the current character cell-size attribute to the value in the order.

Set Character Direction / Push and Set Character Direction

These orders set, or push and set, the value of the current character direction attribute.

Set Character Direction (GSCD)
X'3A'(direction)

Push and Set Character Direction (GPSCD)
X'7A'(direction)

Parameters

direction *(GUCHAR)*
Value for character direction:

All other values are reserved.

X'00' Drawing default
X'01' Left to right
X'02' Top to bottom
X'03' Right to left
X'04' Bottom to top.

Remarks

The value of the current character direction attribute is pushed on to the Segment Call Stack by the Push and Set order only. Both orders then set the value of the current character direction attribute to the value in the order.

Set Character Precision / Push and Set Character Precision

These orders set, or push and set, the value of the current character precision attribute.

Set Character Precision (GSCR)
X'39'(prec)

Push and Set Character Precision (GPSCR)
X'79'(prec)

Parameters

prec *(GUCHAR)*
Value for character-precision attribute:

All other values are reserved.

X'00' Drawing default
X'01' String precision
X'02' Character precision
X'03' Stroke precision.

Remarks

The value of the current character precision attribute is pushed on to the Segment Call Stack by the Push and Set order only. Both orders then set the value of the current character precision attribute to the value in the order.

Set Character Set / Push and Set Character Set

These orders set, or push and set, the value of the current character-set attribute.

Set Character Set (GSCS)
X'38'(lcid)

Push and Set Character Set (GPSCS)
X'78'(lcid)

Parameters

lcid *(GUCHAR)*
Local identifier (lcid) for the character set:

X'00'	Drawing default
X'01'–X'FE'	Lcid for the symbol set
X'FF'	Special character set.

Remarks

The value of the current character-set attribute is pushed on to the Segment Call Stack by the Push and Set order only. Both orders then set the value of the current character-set attribute to the value in the order.

Set Character Shear / Push and Set Character Shear

These orders set, or push and set, the value of the current character shear attribute.

Set Character Shear (GSCH)
X'35'(len, hx, hy)

Push and Set Character Shear (GPSCH)
X'75'(len, hx, hy)

Parameters

len *(GLENGTH1)*
Length of following data.

hx *(GROSOL)*
Dividend of shear ratio.

hy *(GROSOL)*
Divisor of shear ratio.

Remarks

When **hx** and **hy** are both zero, the drawing default is set. The value of the current character shear attribute is pushed on to the Segment Call Stack by the Push and Set order only. Both orders then set the value of the current character shear attribute to the value in the order.

Set Clip Path

This order sets the current clip path.

Set Clip Path (GSCPTH)
X'B4'(len, flags, res, pthId)

Parameters

len *(GLENGTH1)*
Length of following data.

flags
Extra functions:

res *(GBIT1)*
Reserved.

0 Only valid value.

fill *(GBIT1)*
Values:

0 Alternate mode
1 Winding mode.

inter *(GBIT1)*
Values:

0 Set to specified path
1 Set to intersection of specified and current clip path.

res2 *(GBIT5)*
Reserved.

B'00000' Only valid value.

res *(GBIT8)*
Reserved.

X'00' Only valid value.

pthId *(GLONG)*
Path identifier.

0 No clipping.
1 Only valid non zero value.

Set Color / Push and Set Color

These orders set, or push and set, the value of the current color attribute.

Set Color (GSCOL)
X'0A'(col)

Push and Set Color (GPSCOL)
X'4A'(col)

Parameters

col *(GUCHAR)*
Value for color attribute:

X'00' – X'08'	These one-byte values are converted to two-byte values by preceding the value with X'FF'. The resultant is then treated as a two-byte value as defined by Set Extended Color / Push and Set Extended Color.
Other	Reserved values.

Remarks

The value of the current color attribute is pushed on to the Segment Call Stack by the Push and Set order only. Both orders then set the value of the current color attribute to the value in the order.

Set Current Position / Push and Set Current Position

These orders set, or push and set, the value of the current position.

Set Current Position (GSCP)
X'21'(len, p)

Push and Set Current Position (GPSCP)
X'61'(len, p)

Parameters

len *(GLENGTH1)*
Length of following data.

p *(GPOINT)*
Coordinate data.

Remarks

The value of the current position is pushed on to the Segment Call Stack by the Push and Set order only. Both orders then set the value of the current position to the value in the order.

Set Extended Color / Push and Set Extended Color

These orders set, or push and set, the value of the current color attribute.

Set Extended Color (GSECOL)
X'26'(len, color)

Push and Set Extended Color (GPSECOL)
X'66'(len, color)

Parameters

len *(GLENGTH1)*
Length of following data.

2 Only valid value.

color *(GUSHORT)*
Color-table index.

Except for the special values, the values X'0000' through X'nnnn' are allowed keys; that is, as many values as are allowed by the size of the LCT.

Special Values

X'FF00'	Drawing default
X'FF0x'	As value X'000x', where x is in the range 1 through 7
X'FF08'	Reset color, as value X'0000'
Other	All other values are reserved.

Remarks

The value of the current extended color attribute is pushed on to the Segment Call Stack by the Push and Set order only. Both orders then set the value of the current extended color attribute to the value in the order.

Set Fractional Line Width / Push and Set Fractional Line Width

These orders set, or push and set, the value of the current line-width attribute.

Set Fractional Line Width (GSFLW)
X'11'(len, line width)

Push and Set Fractional Line Width (GPSFLW)
X'51'(len, line width)

Parameters

len *(GLENGTH1)*
Length of following data.

line width *(GROUFS)*
Value for the line-width attribute.

The nonzero value is an integral and fractional multiplier of the normal line width:

X'0000'	Drawing default
X'0001' – X'FFFF'	Multiplier of normal line width.

Remarks

The value of the current line-width attribute is pushed on to the Segment Call Stack by the Push and Set order only. Both orders then set the value of the current line-width attribute to the value in the order.

Set Indexed Color / Push and Set Indexed Color

These orders set, or push and set, the value of the current color attribute.

Set Indexed Color (GSICOL)
X'A6'(len, flags, index)

Push and Set Indexed Color (GPSICOL)
X'E6'(len, flags, index)

Parameters

len *(GLENGTH1)*
Length of following data.

flags
Values:

default *(GBIT1)*
Options:

0 Use specified index
1 Use drawing default color.

spec *(GBIT1)*
Options:

0 Use index directly
1 Special Value.

res *(GBIT6)*
Reserved.

000000 Only valid value.

index *(GINDEX3)*
Value for color index.

The value is a direct index into the current color table or a special value.

The table can be the standard table, or one loaded by the user.

The special values are:

1 Black
2 White

4 All ones
5 All zeros.

Remarks

The value of the current color attribute is pushed on to the Segment Call Stack by the Push and Set order only. Both orders then set the value of the current color attribute to the value in the order.

Set Individual Attribute / Push and Set Individual Attribute

These orders set, or push and set, the value of the color, background color, mix, or background mix attribute for the line character, marker, pattern, or image primitive type.

Set Individual Attribute (GSIA)
X'14'(len, atype, ptype, flag1, val)

Push and Set Individual Attribute (GPSIA)
X'54'(len, atype, ptype, flag1, val)

Parameters

len *(GLENGTH1)*
Length of following data.

atype *(GUCHAR)*
Attribute type:

1 Color
2 Background color
3 Mix
4 Background Mix
other All other values are reserved.

ptype *(GUCHAR)*
Primitive type:

1 Line
2 Character
3 Marker
4 Pattern
5 Image
other All other values are reserved.

flag1
Values:

default *(GBIT1)*
Options:

0 Set individual attribute
1 Set default individual attribute.

spec *(GBIT1)*
Options:

0 Use value directly
1 Special Value.

res *(GBIT6)*
Reserved.

000000 Only valid value.

val *(GINDATT)*
Attribute Value.

For colors, the value is a direct index into the current color table or a special value.

The table can be the standard table, or one loaded by the user.

The special values are:

1 Black
2 White
4 All ones
5 All zeros.

Remarks

The value of the current attribute is pushed on to the Segment Call Stack by the Push and Set order only. Both orders then set the value of the individual attribute to the value in the order.

Set Line End / Push and Set Line End

These orders set, or push and set, the value of the current line-end attribute.

Set Line End (GSLE)
X'1A'(lineend)

Push and Set Line End (GPSLE)
X'5A'(lineend)

Parameters

lineend *(GUCHAR)*
Value for the line-end attribute:

X'00' Drawing default
X'01' Flat
X'02' Square
X'03' Round
Other Reserved values.

Remarks

The value of the current line-end attribute is pushed on to the Segment Call Stack by the Push and Set order only. Both orders then set the value of the current line-end attribute to the value in the order.

Set Line Join / Push and Set Line Join

These orders set the value of the current line-join attribute.

Set Line Join (GSLJ)
X'1B'(linejoin)

Push and Set Line Join (GPSLJ)
X'5B'(linejoin)

Parameters

lineJoin *(GUCHAR)*

Value for line-join attribute:

X'00'	Drawing default
X'01'	Bevel
X'02'	Round
X'03'	Miter
Other	Reserved values.

Remarks

The value of the current line-join attribute is pushed on to the Segment Call Stack by the Push and Set order only. Both orders then set the value of the current line-join attribute to the value in the order.

Set Line Type / Push and Set Line Type

These orders set, or push and set, the value of the current line-type attribute.

Set Line Type (GSLT)
X'18'(lineType)

Push and Set Line Type (GPSLT)
X'58'(lineType)

Parameters

lineType *(GUCHAR)*

Value for line-type attribute.

The value is an index into a notational line-type table:

X'00'	Drawing default
X'01'	Dotted line
X'02'	Short dashed line
X'03'	Dash-dot line
X'04'	Double dotted line
X'05'	Long dashed line
X'06'	Dash-double-dot line
X'07'	Solid line
X'08'	Invisible line
Other	Reserved values.

Remarks

The value of the current line-type attribute is pushed on to the Segment Call Stack by the Push and Set order only. Both orders then set the value of the current line-type attribute to the value in the order.

Set Line Width / Push and Set Line Width

These orders set, or push and set, the value of the current line-width attribute to the value specified in the order.

Set Line Width (GSLW)
X'19'(line width)

Push and Set Line Width (GPSLW)
X'59'(line width)

Parameters

line width *(GUCHAR)*
Value for line-width attribute:

X'00'	Drawing default
X'01'–X'FF'	Integral multiplier of normal line width.

Remarks

The value of the current line-width attribute is pushed on to the Segment Call Stack by the Push and Set order only. Both orders then set the value of the current line-width attribute to the value in the order.

Set Marker Cell / Push and Set Marker Cell

These orders set, or push and set, the value of the current marker cell-size attribute.

Set Marker Cell (GSMC)
X'37'(len, cellx, celly, flags, res)

Push and Set Marker Cell (GPSMC)
X'77'(len, cellx, celly, flags, res)

Parameters

len *(GLENGTH1)*
Length of following data.

cellx *(GROSOL)*
X part of marker cell-size attribute.

celly *(GROSOL)*
Y part of marker cell-size attribute.

flags
This is an optional extension.

Values:

notdefault *(GBIT1)*
Options:

0 A cell size of zero sets drawing default
1 A cell size of zero sets to zero.

res *(GBIT7)*
Reserved.

0000000 Only valid value.

res *(GBIT8)*
Reserved.

X'00' Only valid value.

Remarks

The value of the current marker cell-size attribute is pushed on to the Segment Call Stack by the Push and Set order only. Both orders then set the value of the current marker cell-size attribute to the value in the order.

Set Marker Precision / Push and Set Marker Precision

These orders set, or push and set, the value of the current marker-precision attribute.

Set Marker Precision (GSMP)
X'3B'(prec)

Push and Set Marker Precision (GPSMP)
X'7B'(prec)

Parameters

prec *(GUCHAR)*
Value for marker-precision attribute:

X'00' Drawing default
X'01' String precision
X'02' Character precision
X'03' Stroke precision
Other Reserved values.

Remarks

The value of the current marker-precision attribute is pushed on to the Segment Call Stack by the Push and Set order only. Both orders then set the value of the current-marker precision attribute to the value in the order.

Set Marker Set / Push and Set Marker Set

These orders set, or push and set, the value of the current marker symbol-set attribute.

Set Marker Set (GSMS)
X'3C'(lcid)

Push and Set Marker Set (GPSMS)
X'7C'(lcid)

Parameters

lcid *(GUCHAR)*

Local identifier (lcid) for the marker set:

X'00'	Drawing default
X'01' – X'FE'	Lcid for the symbol set
X'FF'	Special marker set.

Remarks

The value of the current marker symbol-set attribute is pushed on to the Segment Call Stack by the Push and Set order only. Both orders then set the value of the current marker symbol-set attribute to the value in the order.

Set Marker Symbol / Push and Set Marker Symbol

These orders set, or push and set, the value of the current marker symbol attribute.

Set Marker Symbol (GSMT)
X'29'(n)

Push and Set Marker Symbol (GPSMT)
X'69'(n)

Parameters

n *(GUCHAR)*

Value of marker symbol code point.

Special marker set

When this is selected (**lcid** = X'FF'), the values are:

X'00'	Drawing default
X'01'	Cross
X'02'	Plus
X'03'	Diamond
X'04'	Square
X'05'	6-point star
X'06'	8-point star
X'07'	Filled diamond
X'08'	Filled square
X'09'	Dot
X'0A'	Small circle
X'40'	Blank
Other	Reserved values.

Marker set

Values are as follows for any other set:

X'00'	Drawing default
X'01' – X'FF'	These are the code points into the current marker set.

Remarks

The value of the current marker symbol attribute is pushed on to the Segment Call Stack by the Push and Set order only. Both orders then set the value of the current marker symbol attribute to the value in the order.

Set Mix / Push and Set Mix

These orders set, or push and set, the value of the current mix attribute.

Set Mix (GSMX)
X'0C'(mode)

Push and Set Mix (GPSMX)
X'4C'(mode)

Parameters

mode *(GUCHAR)*
Mix-mode value:

X'00'	Drawing default
X'01'	OR
X'02'	Overpaint
X'03'	Reserved
X'04'	Exclusive-OR
X'05'	Leave alone
X'06'	AND
X'07'	Subtract
X'08'	Source AND (inverse destination)
X'09'	All zeros
X'0A'	Inverse (source OR destination)
X'0B'	Inverse (source XOR destination)
X'0C'	Inverse destination
X'0D'	Source OR (inverse destination)
X'0E'	Inverse source
X'0F'	(Inverse source) OR destination
X'10'	Inverse (source AND destination)
X'11'	All ones
other	All other values are reserved.

Remarks

The value of the current mix attribute is pushed on to the Segment Call Stack by the Push and Set order only. Both orders then set the value of the current mix attribute to the value in the order.

Set Model Transform / Push and Set Model Transform

These orders set, or push and set, values in the current model transform.

Set Model Transform (GSTM)
X'24'(len, res, flags, mask, mx)

Push and Set Model Transform (GPSTM)
X'64'(len, res, flags, mask, mx)

Parameters

len *(GLENGTH1)*
Length of following data.

res *(GBIT8)*
Reserved.

X'00' Only valid value.

flags
Values:

res *(GBIT6)*
Reserved.

B'000000' Only valid value.

cm *(GBIT2)*
Matrix control bits:

B'00' Unity matrix
B'01' Concatenate after
B'10' Concatenate before
B'11' Overwrite.

mask *(GBIT16)*
Load mask.

mx *(GROSOL*number of bits set on in* **mask***)*
Matrix values.

Remarks

The value of the current model transform is pushed on to the Segment Call Stack by the Push and Set order only. Both orders then set values in the current model transform as specified in the order.

Set Pattern Reference Point / Push and Set Pattern Reference Point

These orders set, or push and set, the value of the current pattern reference-point attribute.

Set Pattern Reference Point (GSPRP)
X'A0'(len, flags, res, pref)

Push and Set Pattern Reference Point (GPSPRP)
X'E0'(len, flags, res, pref)

Parameters

len *(GLENGTH1)*
Length of following data.

flags
Values:

default *(GBIT1)*
Options:

0 Set to specified value
1 Set to the drawing default.

res *(GBIT7)*
Reserved

0000000 Only valid value.

res *(GBIT8)*
Reserved.

X'00' Only valid value.

pref *(GPOINT)*
Coordinate data of the pattern-reference point.

Remarks

The value of the current pattern reference-point attribute is pushed on to the Segment Call Stack by the Push and Set order only. Both orders then set the value of the current reference-point attribute to the value in the order.

Set Pattern Set / Push and Set Pattern Set

These orders set, or push and set, the value of the current pattern symbol-set attribute.

Set Pattern Set (GSPS)
X'08'(lcid)

Push and Set Pattern Set (GPSPS)
X'48'(lcid)

Parameters

lcid *(GUCHAR)*

Local identifier (lcid) for the pattern set:

X'00'	Drawing default
X'01' – X'FE'	Lcid for the symbol set
X'FF'	Special pattern set.

Remarks

The value of the current pattern symbol-set attribute is pushed on to the Segment Call Stack by the Push and Set order only. Both orders then set the value of the current pattern symbol-set attribute to the value in the order.

Set Pattern Symbol / Push and Set Pattern Symbol

These orders set, or push and set, the value of the current pattern-symbol attribute.

Set Pattern Symbol (GSPT)
X'28'(patt)

Push and Set Pattern Symbol (GPSPT)
X'09'(patt)

Parameters

patt *(GUCHAR)*

Value for pattern-symbol attribute.

Special pattern set

When this is selected (**lcid** = X'FF'), the values are:

X'00'	Drawing default
X'01' – X'08'	Density one through density eight (decreasing)
X'09'	Vertical lines
X'0A'	Horizontal lines
X'0B'	Diagonal lines 1 (bottom-left to top-right)
X'0C'	Diagonal lines 2 (bottom-left to top-right)
X'0D'	Diagonal lines 1 (top-left to bottom-right)
X'0E'	Diagonal lines 2 (top-left to bottom-right)
X'0F'	No shading
X'10'	Solid shading
X'40'	Blank
Other	Reserved values.

Pattern set

Values are as follows for any other set:

X'00'	Drawing default
X'01' – X'FF'	These are the code points into the current pattern set.

Remarks

The value of the current pattern-symbol attribute is pushed on to the Segment Call Stack by the Push and Set order only. Both orders then set the value of the current pattern-symbol attribute to the value in the order.

Set Pick Identifier / Push and Set Pick Identifier

These orders set, or push and set, the value of the current pick identifier.

Set Pick Identifier (GSPIK)
X'43'(len, pkId)

Push and Set Pick Identifier (GPSPIK)
X'23'(len, pkId)

Parameters

len *(GLENGTH1)*
Length of following data.

pkId *(GLONG)*
Pick identifier.

Remarks

The value of the current pick identifier is pushed on to the Segment Call Stack by the Push and Set order only. Both orders then set the value of the current pick identifier to the value in the order.

Set Segment Boundary

This order defines the maximum boundaries of the associated root segment.

Set Segment Boundary (GSSB)
X'32'(len, res, mask, bb)

Parameters

len *(GLENGTH1)*
Length of following data.

res *(GBIT8)*
Reserved.

X'00' Only valid value.

mask
Values:

res1 *(GBIT2)*
Reserved.

00 Only valid value.

xl *(GBIT1)*
X left limit.

0 Not included in list of **bb** values
1 Is included in list of **bb** values.

xr *(GBIT1)*
X right limit.

0 Not included in list of **bb** values
1 Is included in list of **bb** values.

yb *(GBIT1)*
Y bottom limit.

0 Not included in list of **bb** values
1 Is included in list of **bb** values.

yt *(GBIT1)*
Y top limit.

0 Not included in list of **bb** values
1 Is included in list of **bb** values.

res2 *(GBIT2)*
Reserved.

00 Only valid value.

bb *(GROSOL*number of bits set on in* **mask***)*
Boundary values.

Remarks

The order is only valid in a root-segment prolog.

Set Segment Characteristics

This order provides the facility to set architected or user-defined characteristics for a segment.

Set Segment Characteristics (GSGCH)
X'04'(len, chid, parms)

Parameters

len *(GLENGTH1)*
Length of following data.

chid *(GUCHAR)*
Identification code for characteristics:

X'00' – X'7F' Reserved for architected characteristics.
X'80' – X'FF' Reserved for user-defined characteristics.

parms *(GSTR)*
Parameters of characteristics.

Remarks

The order is only valid in a root-segment prolog.

Set Stroke Line Width / Push and Set Stroke Line Width

These orders set the current stroke line-width attribute.

Set Stroke Line Width (GSSLW)
X'15'(len, flags, res, stroke width)

Push and Set Stroke Line Width (GPSSLW)
X'55'(len, flags, res, stroke width)

Parameters

len *(GLENGTH1)*
Length of following data.

flags

defit *(GBIT1)*
Values:

0 Set to value
1 Set to drawing default.

res *(GBIT7)*
Reserved.

B'0000000' Only valid value.

res *(GBIT8)*
Reserved.

X'00' Only valid value.

stroke width *(GROSOL)*
Value for stroke width.

Set Viewing Transform

This order sets the current viewing transform.

Set Viewing Transform (GSTV)
X'31'(len, res, flags, mask, mx)

Parameters

len *(GLENGTH1)*
Length of following data.

res *(GBIT8)*
Reserved.

0 Only valid value.

flags
Values:

res1 *(GBIT5)*
Reserved.

00000 Only valid value.

control *(GBIT1)*
Values:

0 Concatenate before drawing default
1 Concatenate before drawing the current viewing transform.

res2 *(GBIT2)*
Reserved.

00 Only valid value.

mask *(GBIT16)*
Load mask.

mx *(GROSOL*number of bits set on in* **mask**)
Matrix values.

Set Viewing Window / Push and Set Viewing Window

These orders set, or push and set, the current viewing window.

Set Viewing Window (GSVW)
X'27'(len, flag, mask, ww)

Push and Set Viewing Window (GPSVW)
X'67'(len, flag, mask, ww)

Parameters

len *(GLENGTH1)*
Length of following data.

flag
Values:

replace *(GBIT1)*
Values:

0 Intersect with current window
1 Replace current with new window.

res *(GBIT7)*
Reserved.

0000000 Only valid value.

mask
Values:

res1 *(GBIT2)*
Reserved.

00 Only valid value.

xl *(GBIT1)*
X left limit.

0 Not included in list of **ww** values
1 Is included in list of **ww** values.

xr *(GBIT1)*
X right limit.

0 Not included in list of **ww** values
1 Is included in list of **ww** values.

yb *(GBIT1)*
Y bottom limit.

0 Not included in list of **ww** values
1 Is included in list of **ww** values.

yt *(GBIT1)*
Y top limit.

0 Not included in list of **ww** values
1 Is included in list of **ww** values.

res2 *(GBIT2)*
Reserved value.

00 Only valid value.

ww *(GROSOL*number of bits set on in* **mask***)*
Window values.

Remarks

The value of the current viewing window is pushed on to the Segment Call Stack by the Push and Set order only. Both orders then set the current viewing window using the values in the order.

Sharp Fillet at Given Position / Sharp Fillet at Current Position

These orders generate a curve that starts at the given or current position, and uses points P1 and P2, together with sharpness specification S1.

Sharp Fillet at Given Position (GSFLT)
X'E4'(len, p0, p1, p2, p3,p4, pn-1,pn, s1, s2, sn/2)

Sharp Fillet at Current Position (GCSFLT)
X'A4'(len, p1, p2, p3,p4, pn-1,pn, s1, s2, sn/2)

Parameters

len *(GLENGTH1)*
Length of following data.

p0 *(GPOINT)*
Coordinate data of first curve start.

This parameter is only present in a Sharp Fillet at Given Position order.

p1 *(GPOINT)*
Coordinate data of control point.

p2 *(GPOINT)*
Coordinate data of first curve end.

p3,p4 *(GPOINT)*
Coordinate data for second curve.

pn-1,pn *(GPOINT)*
Coordinate data for last curve.

s1 *(GROF)*
Sharpness specification of first curve.

s2 *(GROF)*
Sharpness specification of second curve.

sn/2 *(GROF)*
Sharpness specification of last curve.

Remarks

Further points are used in groups of two to form a polycurve.

Chapter 28. Code Pages

The initialization file contains country information relating to date, time, and numeric formats. It does not contain code-page information; this is obtained from CONFIG.SYS.

Applications start with the default code page. This default code page is set when Operating System/2 (OS/2) Version 1.2 is first installed and can be changed subsequently either by reinstalling OS/2 or by editing the 'COUNTRY=' statement in the file CONFIG.SYS.

The code page of a VIO-windowed application inherits the code page of the process that created it; its code page changes subsequently when any process of the same application issues a DosSetCp or VioSetCp.

A GPI or AVIO presentation space also inherits the code page of the process that created it but a change in code page occurs only when the process itself issues a GpiSetCp.

Full Screen VIO Applications

Full screen VIO applications may use one of the two code pages set in CONFIG.SYS. These code pages may be any of the following:

	Char Set	Code Page
Canadian-French	993	863
Multilingual	980	850
Nordic	995	865
Portuguese	990	860
U.S. (IBM PC)	919	437

OS/2 Version 1.2 provides seven code-page switching functions that refer to code pages defined by the 'CODEPAGE=' statement in the file CONFIG.SYS.

VioSetCp Sets the code page for AVIO.
VioGetCp Queries the code page for AVIO.
KbdSetCp Sets the code page for KBD.
KbdGetCp Queries the code page for KBD.
DosGetCp Get code page of current process.
DosSetCp Set code page for current process.
DosSetProcCp Set code page for current process (keyboard and display are not changed).

Windowed PM Applications

Windowed Presentation Manager applications allow the code page functions to use any of the supported ASCII code pages. The following EBCDIC code pages, based on character set 697, are also recommended:

	Char Set	Code Page
Austrian/German	697	273
Belgian	697	500
Danish/Norwegian	697	277
Finnish/Swedish	697	278
French	697	297
International	697	500
Italian	697	280
Portuguese	697	037
Spanish	697	284
U.K.-English	697	285
U.S.-English	697	037

Programming Note: Code pages 274 (Belgian) and 282 (Portuguese) can be used to provide access to old data.

OS/2 Version 1.2 provides additional code-page setting and query functions for the supported ASCII and EBCDIC code pages. These functions work independently of the file CONFIG.SYS.

GpiSetCp	Sets the code page for GPI.
GpiQueryCp	Queries the code page for GPI.
GpiCreateLogFont	Creates fonts in a code page.
VioCreateLogFont	Creates AVIO fonts in a code page.
VioSetCp	Sets the code page for AVIO.
VioGetCp	Gets the code page for AVIO.
WinSetCp	Sets the code page for a message queue.
WinQueryCp	Queries the code page for a message queue.

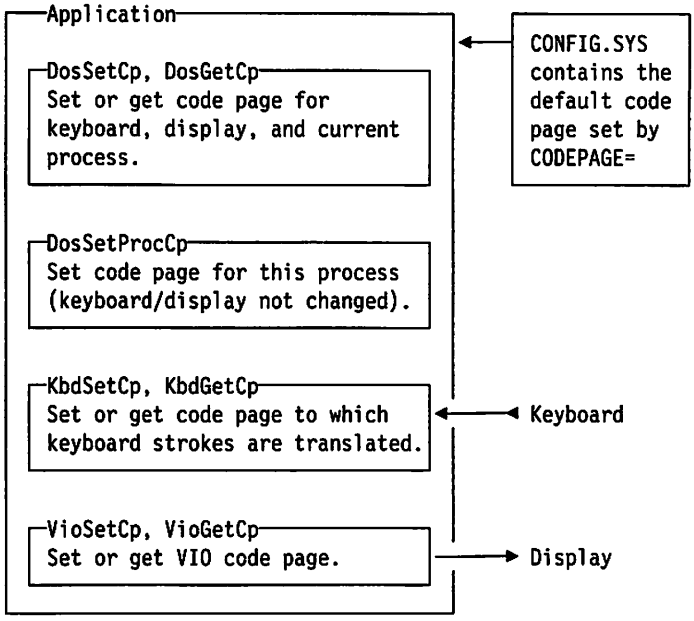
WinQueryCpList creates a list of code pages supported by OS/2 Version 1.2.

Text entered in a dialog box is supplied to the application in the code page of the queue ('queue code page').

If possible the code page of a resource, for example, a menu or dialog box, should match the code page of the queue. In general, code page 850 is the best choice for both an application and its resources.

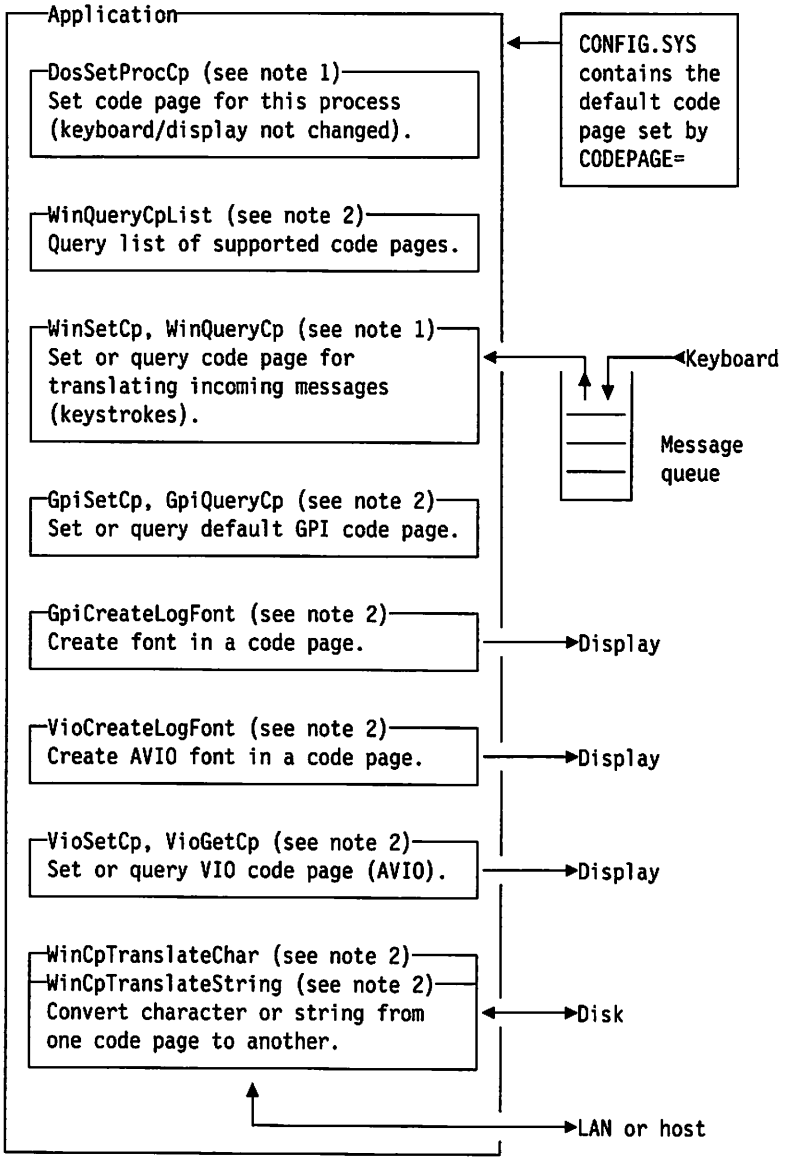
Applications should be able to process data from a variety of sources. Because code page 850 contains most of the characters in other supported code pages, a queue code page of 850 is usually the best choice.

OS/2 Version 1.2 Code Page Options for VIO and PM Applications



Note: The chosen code-page must be one of the two ASCII code pages specified in CONFIG.SYS (CODEPAGE= statement).

Figure 28-1. OS/2 Version 1.2 Code Page Options for VIO Applications



Note 1: Either of the two ASCII code pages specified in CONFIG.SYS.
Note 2: Any supported ASCII or EBCDIC code page as reported by
WinQueryCpList.

Figure 28-2. OS/2 Version 1.2 Code Page Options for PM Applications

OS/2 Version 1.2 Font Support For Multiple Code Pages

OS/2 Version 1.2 supports multiple code pages for text input and output. A single font resource is used to support all the code pages. The following section describes how this function is provided and describes the font resource format.

Font Code Page Functions

Many of the characters required by each code page are common; for example, the first 128 characters of all the ASCII code pages are identical. This set of characters is called the Universal Glyph List (UGL). A code page is simply a set of pointers into the UGL.

As the characters in every font are in the same order, only one set of code-page translation tables is necessary.

Font Layout

The ordering of characters in fonts is based on that of code-page 850, with additional characters added beyond the 256th. These are shown in the order they occur in the multi-code-page font, starting at character number 256:

Index Number	Colloquial Name
256	Pesetas sign
257	Left-hand not sign
258	Double line join single vertical
259	Single line join double vertical
260	Single line top right corner double
261	Double line top right corner single
262	Single line bottom right corner double
263	Double line bottom right corner single
264	Single vertical join double line
265	Double vertical join single line
266	Double horizontal join single line above
267	Single horizontal join double line above
268	Double horizontal join single line below
269	Single horizontal join double line below
270	Double line bottom left corner single
271	Single line bottom left corner double
272	Single line top left corner double
273	Double line top left corner single
274	Double vertical cross single
275	Single vertical cross double
276	Left hand half-block
277	Right hand half-block
278	Greek alpha lowercase
279	Greek gamma uppercase
280	Greek pi lowercase
281	Greek sigma uppercase
282	Greek sigma lowercase
283	Greek tau lowercase
284	Greek phi uppercase
285	Greek theta uppercase
286	Greek omega uppercase
287	Greek delta lowercase
288	Infinity sign
289	Greek phi lowercase
290	Greek epsilon lowercase
291	Mathematical intersection sign
292	Mathematical equivalence sign
293	Mathematical greater than or equals sign
294	Mathematical less than or equals sign
295	Mathematical integral sign top half

Index Number	Colloquial Name
296	Mathematical integral sign bottom half
297	Mathematical approximately equals sign
298	Mathematical product dot
299	Mathematical square root sign
300	Superscript small n
301	Macron
302	Brev
303	Overdot accent
304	Overcircle accent
305	Double acute accent (Hungarian umlaut)
306	Ogenek
307	Caron
308	Left single quote
309	Right single quote
310	Left double quote
311	Right double quote
312	En dash
313	Em dash
314	ASCII circumflex
315	ASCII tilde
316	German low single quote
317	Left Lower Double Quotes
318	Ellipsis (...)
319	Dagger Footnote indicator
320	Double Dagger Footnote indicator
321	Circumflex Accent (over small alpha)
322	Permille symbol
323	S Caron Capital
324	French single open quote
325	OE Ligature Capital
326	Tilde accent (over small alpha)
327	Trademark Symbol
328	s Caron Small
329	French single close quote
330	oe Ligature Small
331	Y Diaeresis Capital

ASCII Code Pages

1 → 2 ↓	0-	1-	2-	3-	4-	5-	6-	7-		8-	9-	A-	B-	C-	D-	E-	F-
-0		►		0	@	P	`	p		Ç	É	á	▒	└	╨	α	≡
-1	☺	◄	!	1	A	Q	a	q		ü	æ	í	▒	┴	╤	β	±
-2	☻	↕	"	2	B	R	b	r		é	Æ	ó	▓	┬	╥	Γ	≥
-3	♥	!!	#	3	C	S	c	s		â	ô	ú	│	├	╙	π	≤
-4	♦	¶	$	4	D	T	d	t		ä	ö	ñ	┤	─	╘	Σ	⌠
-5	♣	§	%	5	E	U	e	u		à	ò	Ñ	╡	┼	╒	σ	⌡
-6	♠	▬	&	6	F	V	f	v		å	û	ª	╢	╞	╓	µ	÷
-7	•	↨	'	7	G	W	g	w		ç	ù	º	╖	╟	╫	τ	≈
-8	◘	↑	(	8	H	X	h	x		ê	ÿ	¿	╕	╚	╪	Φ	°
-9	○	↓	)	9	I	Y	i	y		ë	Ö	⌐	╣	╔	┘	Θ	•
-A	◙	→	*	:	J	Z	j	z		è	Ü	¬	║	╩	┌	Ω	·
-B	♂	←	+	;	K	[	k	{		ï	¢	½	╗	╦	■	δ	√
-C	♀	∟	,	<	L	\	l	\|		î	£	¼	╝	╠	▬	∞	ⁿ
-D	♪	↔	-	=	M	]	m	}		ì	¥	¡	╜	═	▌	φ	²
-E	♫	▲	.	>	N	^	n	~		Ä	Pt	«	╛	╬	▐	ε	■
-F	☼	▼	/	?	O	_	o	⌂		Å	ƒ	»	┐	╧	▀	∩	

Figure 28-3. US-English: ASCII Code Page 437

1 → 2 ↓	0-	1-	2-	3-	4-	5-	6-	7-		8-	9-	A-	B-	C-	D-	E-	F-
-0		►		0	@	P	`	p		Ç	É	á	▒	└	ð	Ó	-
-1	☺	◄	!	1	A	Q	a	q		ü	æ	í	▒	┴	Ð	β	±
-2	☻	↕	"	2	B	R	b	r		é	Æ	ó	▒	┬	Ê	Ô	=
-3	♥	!!	#	3	C	S	c	s		â	ô	ú	▒	├	Ë	Ò	¾
-4	♦	¶	$	4	D	T	d	t		ä	ö	ñ	▒	─	È	õ	¶
-5	♣	§	%	5	E	U	e	u		à	ò	Ñ	Á	┼	ı	Õ	§
-6	♠	▬	&	6	F	V	f	v		å	û	ª	Â	ã	Í	µ	÷
-7	•	↨	'	7	G	W	g	w		ç	ù	º	À	Ã	Î	þ	¸
-8	◘	↑	(	8	H	X	h	x		ê	ÿ	¿	©	╚	Ï	Þ	°
-9	○	↓	)	9	I	Y	i	y		ë	Ö	®	╣	╔	┘	Ú	··
-A	◙	→	*	:	J	Z	j	z		è	Ü	¬	║	╩	┌	Û	·
-B	♂	←	+	;	K	[	k	{		ï	ø	½	╗	╦	▉	Ù	¹
-C	♀	∟	,	<	L	\	l	\|		î	£	¼	╝	╠	▬	ý	³
-D	♪	↔	-	=	M	]	m	}		ì	Ø	¡	¢	═	¦	Ý	²
-E	♫	▲	.	>	N	^	n	~		Ä	×	«	¥	╬	Ì	¯	■
-F	☼	▼	/	?	O	_	o	⌂		Å	ƒ	»	┐	¤	▬	´	

Figure 28-4. Multilingual: ASCII Code Page 850

1 → 2 ↓	0-	1-	2-	3-	4-	5-	6-	7-		8-	9-	A-	B-	C-	D-	E-	F-
-0		►		0	@	P	`	p		Ç	É	á	▒	└	╨	α	≡
-1	☺	◄	!	1	A	Q	a	q		ü	À	í	▒	┴	╤	β	±
-2	☻	↕	"	2	B	R	b	r		é	È	ó	▒	┬	╥	Γ	≥
-3	♥	!!	#	3	C	S	c	s		â	ô	ú	▒	├	╙	π	≤
-4	♦	¶	$	4	D	T	d	t		ã	õ	ñ	▒	─	╘	Σ	⌠
-5	♣	§	%	5	E	U	e	u		à	ò	Ñ	▒	┼	╒	σ	⌡
-6	♠	▬	&	6	F	V	f	v		Á	Ú	ª	▒	╞	╓	µ	÷
-7	•	↨	'	7	G	W	g	w		ç	ù	º	▒	╟	╫	τ	≈
-8	◘	↑	(	8	H	X	h	x		ê	Ì	¿	▒	╚	╪	Φ	°
-9	○	↓	)	9	I	Y	i	y		Ê	Õ	Ò	▒	╔	┘	Θ	•
-A	◙	→	*	:	J	Z	j	z		è	Ù	¬	▒	╩	┌	Ω	·
-B	♂	←	+	;	K	[	k	{		Ì	¢	½	▒	╦	▉	δ	√
-C	♀	∟	,	<	L	\	l	\|		Ô	£	¼	▒	╠	▬	∞	ⁿ
-D	♪	↔	-	=	M	]	m	}		ì	Ù	¡	▒	═	▌	φ	²
-E	♫	▲	.	>	N	^	n	~		Ã	Pt	«	▒	╬	▐	ε	■
-F	☼	▼	/	?	O	_	o	⌂		Â	Ó	»	▒	╧	▬	∩	

Figure 28-5. Portuguese: ASCII Code Page 860

1 → 2 ↓	0-	1-	2-	3-	4-	5-	6-	7-		8-	9-	A-	B-	C-	D-	E-	F-
-0		►		0	@	P	`	p		Ç	É	¦	⣿	└	╨	α	≡
-1	☺	◄	!	1	A	Q	a	q		ü	È	´	▒	┴	╤	β	±
-2	☻	↕	"	2	B	R	b	r		é	Ê	ó	▓	┬	╥	Γ	≥
-3	♥	!!	#	3	C	S	c	s		â	ô	ú	│	├	╙	π	≤
-4	♦	¶	$	4	D	T	d	t		Â	Ë	¨	┤	─	╘	Σ	⌠
-5	♣	§	%	5	E	U	e	u		à	Ï	˙	╡	┼	╒	σ	⌡
-6	♠	‒	&	6	F	V	f	v		¶	û	³	╢	╞	╓	µ	÷
-7	•	↨	'	7	G	W	g	w		ç	ù	˙	╖	╟	╫	τ	≈
-8	◘	↑	(	8	H	X	h	x		ê	¤	Î	╕	╚	╪	Φ	°
-9	○	↓	)	9	I	Y	i	y		ë	Ô	⌐	╣	╔	┘	Θ	•
-A	◙	→	*	:	J	Z	j	z		è	Ù	¬	║	╩	┌	Ω	·
-B	♂	←	+	;	K	[	k	{		ï	¢	½	╗	╦	■	δ	√
-C	♀	∟	,	<	L	\	l	\|		î	£	¼	╝	╠	▬	∞	ⁿ
-D	♪	↔	-	=	M	]	m	}		=	Ù	¾	╜	═	▌	φ	²
-E	♫	▲	.	>	N	^	n	~		À	Û	«	╛	╬	▐	ε	■
-F	☼	▼	/	?	O	_	o	⌂		§	ƒ	»	┐	╧	▬	∩	

Figure 28-6. Canadian-French: ASCII Code Page 863

1 → 2 ↓	0-	1-	2-	3-	4-	5-	6-	7-		8-	9-	A-	B-	C-	D-	E-	F-
-0		►		0	@	P	`	p		Ç	É	á	⣿	└	╨	α	≡
-1	☺	◄	!	1	A	Q	a	q		ü	æ	í	▒	┴	╤	β	±
-2	☻	↕	"	2	B	R	b	r		é	Æ	ó	⣿	┬	╥	Γ	≥
-3	♥	!!	#	3	C	S	c	s		â	ô	ú	⎸	├	╙	π	≤
-4	♦	¶	$	4	D	T	d	t		ä	ö	ñ	┤	─	╘	Σ	⌠
-5	♣	§	%	5	E	U	e	u		à	ò	Ñ	╡	┼	╒	σ	⌡
-6	♠	▬	&	6	F	V	f	v		å	û	ª	╢	╞	╓	µ	÷
-7	•	↨	'	7	G	W	g	w		ç	ù	º	╖	╟	╫	τ	≈
-8	◘	↑	(	8	H	X	h	x		ê	ÿ	¿	╕	╚	╪	Φ	°
-9	○	↓	)	9	I	Y	i	y		ë	Ö	⌐	╣	╔	┘	Θ	•
-A	◙	→	*	:	J	Z	j	z		è	Ü	¬	║	╩	┌	Ω	·
-B	♂	←	+	;	K	[	k	{		ï	ø	½	╗	╦	▉	δ	√
-C	♀	∟	,	<	L	\	l	\|		î	£	¼	╝	╠	▬	∞	ⁿ
-D	♪	↔	-	=	M	]	m	}		ì	Ø	¡	╜	═	▌	φ	²
-E	♫	▲	.	>	N	^	n	~		Ä	Pt	«	╛	╬	▐	ε	■
-F	☼	▼	/	?	O	_	o	⌂		Å	ƒ	¤	┐	╧	▀	∩	

Figure 28-7. Norwegian: ASCII Code Page 865

EBCDIC Code Pages

1 → 2 ↓	4-	5-	6-	7-	8-	9-	A-	B-	C-	D-	E-	F-
-0		&	-	ø	Ø	°	µ	^	{	}	\	0
-1		é	/	É	a	j	~	£	A	J	÷	1
-2	â	ê	Â	Ê	b	k	s	¥	B	K	S	2
-3	ä	ë	Ä	Ë	c	l	t	•	C	L	T	3
-4	à	è	À	È	d	m	u	©	D	M	U	4
-5	á	í	Á	Í	e	n	v	§	E	N	V	5
-6	ã	î	Ã	Î	f	o	w	¶	F	O	W	6
-7	å	ï	Å	Ï	g	p	x	¼	G	P	X	7
-8	ç	ì	Ç	Ì	h	q	y	½	H	Q	Y	8
-9	ñ	ß	Ñ	`	i	r	z	¾	I	R	Z	9
-A	¢	!	¦	:	«	ª	¡	[	-	¹	²	³
-B	.	$	,	#	»	º	¿	]	ô	û	Ô	Û
-C	<	*	%	@	ð	æ	Ð	¯	ö	ü	Ö	Ü
-D	(	)	_	'	ý	¸	Ý	¨	ò	ù	Ò	Ù
-E	+	;	>	=	þ	Æ	Þ	´	ó	ú	Ó	Ú
-F	\|	¬	?	"	±	¤	®	×	õ	ÿ	Õ	EO

Figure 28-8. US-English: EBCDIC Code Page 037

1 → 2 ↓	4-	5-	6-	7-	8-	9-	A-	B-	C-	D-	E-	F-
-0		&	-	ø	Ø	°	µ	¢	ä	ü	Ö	0
-1		é	/	É	a	j	ß	£	A	J	÷	1
-2	â	ê	Â	Ê	b	k	s	¥	B	K	S	2
-3	{	ë	[	Ë	c	l	t	•	C	L	T	3
-4	à	è	À	È	d	m	u	©	D	M	U	4
-5	á	í	Á	Í	e	n	v	@	E	N	V	5
-6	ã	î	Ã	Î	f	o	w	¶	F	O	W	6
-7	å	ï	Å	Ï	g	p	x	¼	G	P	X	7
-8	ç	ì	Ç	Ì	h	q	y	½	H	Q	Y	8
-9	ñ	~	Ñ	`	i	r	z	¾	I	R	Z	9
-A	Ä	Ü	ö	:	«	ª	¡	¬	-	¹	²	³
-B	.	$	,	#	»	º	¿	\|	ô	û	Ô	Û
-C	<	*	%	§	ð	æ	Ð	¯	¦	}	\	]
-D	(	)	_	'	ý	¸	Ý	¨	ò	ù	Ò	Ù
-E	+	;	>	=	þ	Æ	Þ	´	ó	ú	Ó	Ú
-F	!	^	?	"	±	¤	®	×	õ	ÿ	Õ	EO

Figure 28-9. Austrian/German: EBCDIC Code Page 273

1 → 2 ↓	4-	5-	6-	7-	8-	9-	A-	B-	C-	D-	E-	F-
-0		&	-	ø	Ø	°	µ	¢	é	è	ç	0
-1		{	/	É	a	j	¨	£	A	J	÷	1
-2	â	ê	Â	Ê	b	k	s	¥	B	K	S	2
-3	ä	ë	Ä	Ë	c	l	t	•	C	L	T	3
-4	@	}	À	È	d	m	u	©	D	M	U	4
-5	á	í	Á	Í	e	n	v	§	E	N	V	5
-6	ã	î	Ã	Î	f	o	w	¶	F	O	W	6
-7	å	ï	Å	Ï	g	p	x	¼	G	P	X	7
-8	\	ì	Ç	Ì	h	q	y	½	H	Q	Y	8
-9	ñ	ß	Ñ	`	i	r	z	¾	I	R	Z	9
-A	[	]	ù	:	«	ª	¡	¬	-	¹	²	³
-B	.	$	,	#	»	º	¿	\|	ô	û	Ô	Û
-C	<	*	%	à	ð	æ	Ð	¯	ö	ü	Ö	Ü
-D	(	)	_	'	ý	¸	Ý	~	ò	¦	Ò	Ù
-E	+	;	>	=	þ	Æ	Þ	´	ó	ú	Ó	Ú
-F	!	^	?	"	±	¤	®	×	õ	ÿ	Õ	EO

Figure 28-10. Belgian: EBCDIC Code Page 274 (supported for migration purposes)

1 → 2 ↓	4-	5-	6-	7-	8-	9-	A-	B-	C-	D-	E-	F-
-0		&	-	¦	@	°	µ	¢	æ	å	\	0
-1		é	/	É	a	j	ü	£	A	J	÷	1
-2	â	ê	Â	Ê	b	k	s	¥	B	K	S	2
-3	ä	ë	Ä	Ë	c	l	t	•	C	L	T	3
-4	à	è	À	È	d	m	u	©	D	M	U	4
-5	á	í	Á	Í	e	n	v	§	E	N	V	5
-6	ã	î	Ã	Î	f	o	w	¶	F	O	W	6
-7	}	ï	$	Ï	g	p	x	¼	G	P	X	7
-8	ç	ì	Ç	Ì	h	q	y	½	H	Q	Y	8
-9	ñ	ß	Ñ	`	i	r	z	¾	I	R	Z	9
-A	#	¤	ø	:	«	ª	¡	¬	-	¹	²	³
-B	.	Å	,	Æ	»	º	¿	\|	ô	û	Ô	Û
-C	<	*	%	Ø	ð	{	Ð	¯	ö	~	Ö	Ü
-D	(	)	_	'	ý	¸	Ý	¨	ò	ù	Ò	Ù
-E	+	;	>	=	þ	[	Þ	´	ó	ú	Ó	Ú
-F	!	^	?	"	±	]	®	×	õ	ÿ	Õ	EO

Figure 28-11. Danish/Norwegian: EBCDIC Code Page 277

1 → 2 ↓	4-	5-	6-	7-	8-	9-	A-	B-	C-	D-	E-	F-
-0		&	-	ø	Ø	°	µ	¢	ä	å	É	0
-1		`	/	\	a	j	ü	£	A	J	÷	1
-2	â	ê	Â	Ê	b	k	s	¥	B	K	S	2
-3	{	ë	#	Ë	c	l	t	•	C	L	T	3
-4	à	è	À	È	d	m	u	©	D	M	U	4
-5	á	í	Á	Í	e	n	v	[	E	N	V	5
-6	ã	î	Ã	Î	f	o	w	¶	F	O	W	6
-7	}	ï	$	Ï	g	p	x	¼	G	P	X	7
-8	ç	ì	Ç	Ì	h	q	y	½	H	Q	Y	8
-9	ñ	ß	Ñ	é	i	r	z	¾	I	R	Z	9
-A	§	¤	ö	:	«	ª	¡	¬	-	¹	²	³
-B	.	Å	,	Ä	»	º	¿	\|	ô	û	Ô	Û
-C	<	*	%	Ö	ð	æ	Ð	¯	¦	~	@	Ü
-D	(	)	_	'	ý	¸	Ý	¨	ò	ù	Ò	Ù
-E	+	;	>	=	þ	Æ	Þ	´	ó	ú	Ó	Ú
-F	!	^	?	"	±	]	®	×	õ	ÿ	Õ	EO

Figure 28-12. Finnish/Swedish: EBCDIC Code Page 278

1 → 2 ↓	4-	5-	6-	7-	8-	9-	A-	B-	C-	D-	E-	F-
-0		&	-	ø	Ø	[	µ	¢	à	è	ç	0
-1		]	/	É	a	j	ì	#	A	J	÷	1
-2	â	ê	Â	Ê	b	k	s	¥	B	K	S	2
-3	ä	ë	Ä	Ë	c	l	t	•	C	L	T	3
-4	{	}	À	È	d	m	u	©	D	M	U	4
-5	á	í	Á	Í	e	n	v	@	E	N	V	5
-6	ã	î	Ã	Î	f	o	w	¶	F	O	W	6
-7	å	ï	Å	Ï	g	p	x	¼	G	P	X	7
-8	\	~	Ç	Ì	h	q	y	½	H	Q	Y	8
-9	ñ	ß	Ñ	ù	i	r	z	¾	I	R	Z	9
-A	°	é	ò	:	«	ª	¡	¬	-	¹	²	³
-B	.	$	,	£	»	º	¿	\|	ô	û	Ô	Û
-C	<	*	%	§	ð	æ	Ð	¯	ö	ü	Ö	Ü
-D	(	)	_	'	ý	¸	Ý	¨	¦	`	Ò	Ù
-E	+	;	>	=	þ	Æ	Þ	´	ó	ú	Ó	Ú
-F	!	^	?	"	±	¤	®	×	õ	ÿ	Õ	EO

Figure 28-13. Italian: EBCDIC Code Page 280

1 → 2 ↓	4-	5-	6-	7-	8-	9-	A-	B-	C-	D-	E-	F-
-0		&	-	ø	Ø	°	µ	¢	ã	´	Ç	0
-1		é	/	É	a	j	ç	£	A	J	÷	1
-2	â	ê	Â	Ê	b	k	s	¥	B	K	S	2
-3	ä	ë	Ä	Ë	c	l	t	•	C	L	T	3
-4	à	è	À	È	d	m	u	©	D	M	U	4
-5	á	í	Á	Í	e	n	v	§	E	N	V	5
-6	{	î	#	Î	f	o	w	¶	F	O	W	6
-7	å	ï	Å	Ï	g	p	x	¼	G	P	X	7
-8	~	ì	\	Ì	h	q	y	½	H	Q	Y	8
-9	ñ	ß	Ñ	`	i	r	z	¾	I	R	Z	9
-A	[	]	õ	:	«	ª	¡	¬	-	¹	²	³
-B	.	$	,	Ã	»	º	¿	\|	ô	û	Ô	Û
-C	<	*	%	Õ	ð	æ	Ð	¯	ö	ü	Ö	Ü
-D	(	)	_	'	ý	¸	Ý	¨	ò	ù	Ò	Ù
-E	+	;	>	=	þ	Æ	Þ	}	ó	ú	Ó	Ú
-F	!	^	?	"	±	¤	®	×	¦	ÿ	@	EO

Figure 28-14. Portuguese: EBCDIC Code Page 282 (supported for migration purposes)

1 → 2 ↓	4-	5-	6-	7-	8-	9-	A-	B-	C-	D-	E-	F-
-0		&	-	ø	Ø	°	µ	¢	{	}	\	0
-1		é	/	É	a	j	¨	£	A	J	÷	1
-2	â	ê	Â	Ê	b	k	s	¥	B	K	S	2
-3	ä	ë	Ä	Ë	c	l	t	•	C	L	T	3
-4	à	è	À	È	d	m	u	©	D	M	U	4
-5	á	í	Á	Í	e	n	v	§	E	N	V	5
-6	ã	î	Ã	Î	f	o	w	¶	F	O	W	6
-7	å	ï	Å	Ï	g	p	x	¼	G	P	X	7
-8	ç	ì	Ç	Ì	h	q	y	½	H	Q	Y	8
-9	¦	ß	#	`	i	r	z	¾	I	R	Z	9
-A	[	]	ñ	:	«	ª	¡	^	-	¹	²	³
-B	.	$	,	Ñ	»	º	¿	!	ô	û	Ô	Û
-C	<	*	%	@	ð	æ	Ð	¯	ö	ü	Ö	Ü
-D	(	)	_	'	ý	¸	Ý	~	ò	ù	Ò	Ù
-E	+	;	>	=	þ	Æ	Þ	´	ó	ú	Ó	Ú
-F	\|	¬	?	"	±	¤	®	×	õ	ÿ	Õ	EO

Figure 28-15. Spanish: EBCDIC Code Page 284

1 → 2 ↓	4-	5-	6-	7-	8-	9-	A-	B-	C-	D-	E-	F-
-0		&	-	ø	Ø	°	µ	¢	{	}	\	0
-1		é	/	É	a	j	¯	[	A	J	÷	1
-2	â	ê	Â	Ê	b	k	s	¥	B	K	S	2
-3	ä	ë	Ä	Ë	c	l	t	•	C	L	T	3
-4	à	è	À	È	d	m	u	©	D	M	U	4
-5	á	í	Á	Í	e	n	v	§	E	N	V	5
-6	ã	î	Ã	Î	f	o	w	¶	F	O	W	6
-7	å	ï	Å	Ï	g	p	x	¼	G	P	X	7
-8	ç	ì	Ç	Ì	h	q	y	½	H	Q	Y	8
-9	ñ	ß	Ñ	`	i	r	z	¾	I	R	Z	9
-A	$	!	¦	:	«	ª	¡	^	-	¹	²	³
-B	.	£	,	#	»	º	¿	]	ô	û	Ô	Û
-C	<	*	%	@	ð	æ	Ð	~	ö	ü	Ö	Ü
-D	(	)	_	'	ý	¸	Ý	¨	ò	ù	Ò	Ù
-E	+	;	>	=	þ	Æ	Þ	´	ó	ú	Ó	Ú
-F	\|	¬	?	"	±	¤	®	×	õ	ÿ	Õ	EO

Figure 28-16. UK-English: EBCDIC Code Page 285

1 → 2 ↓	4-	5-	6-	7-	8-	9-	A-	B-	C-	D-	E-	F-
-0		&	-	ø	Ø	[	`	¢	é	è	ç	0
-1		{	/	É	a	j	¨	#	A	J	÷	1
-2	â	ê	Â	Ê	b	k	s	¥	B	K	S	2
-3	ä	ë	Ä	Ë	c	l	t	•	C	L	T	3
-4	@	}	À	È	d	m	u	©	D	M	U	4
-5	á	í	Á	Í	e	n	v	]	E	N	V	5
-6	ã	î	Ã	Î	f	o	w	¶	F	O	W	6
-7	å	ï	Å	Ï	g	p	x	¼	G	P	X	7
-8	\	ì	Ç	Ì	h	q	y	½	H	Q	Y	8
-9	ñ	ß	Ñ	µ	i	r	z	¾	I	R	Z	9
-A	°	§	ù	:	«	ª	¡	¬	-	¹	²	³
-B	.	$	,	£	»	º	¿	\|	ô	û	Ô	Û
-C	<	*	%	à	ð	æ	Ð	¯	ö	ü	Ö	Ü
-D	(	)	_	'	ý	¸	Ý	~	ò	¦	Ò	Ù
-E	+	;	>	=	þ	Æ	Þ	´	ó	ú	Ó	Ú
-F	!	^	?	"	±	¤	®	×	õ	ÿ	Õ	EO

Figure 28-17. French: EBCDIC Code Page 297

1 → 2 ↓	4-	5-	6-	7-	8-	9-	A-	B-	C-	D-	E-	F-
-0		&	-	ø	Ø	°	µ	¢	{	}	\	0
-1		é	/	É	a	j	~	£	A	J	÷	1
-2	â	ê	Â	Ê	b	k	s	¥	B	K	S	2
-3	ä	ë	Ä	Ë	c	l	t	•	C	L	T	3
-4	à	è	À	È	d	m	u	©	D	M	U	4
-5	á	í	Á	Í	e	n	v	§	E	N	V	5
-6	ã	î	Ã	Î	f	o	w	¶	F	O	W	6
-7	å	ï	Å	Ï	g	p	x	¼	G	P	X	7
-8	ç	ì	Ç	Ì	h	q	y	½	H	Q	Y	8
-9	ñ	ß	Ñ	`	i	r	z	¾	I	R	Z	9
-A	[	]	¦	:	«	ª	¡	¬	-	¹	²	³
-B	.	$	,	#	»	º	¿	\|	ô	û	Ô	Û
-C	<	*	%	@	ð	æ	Ð	¯	ö	ü	Ö	Ü
-D	(	)	_	'	ý	¸	Ý	¨	ò	ù	Ò	Ù
-E	+	;	>	=	þ	Æ	Þ	´	ó	ú	Ó	Ú
-F	!	^	?	"	±	¤	®	×	õ	ÿ	Õ	EO

Figure 28-18. International: EBCDIC Code Page 500

DBCS Support

The Presentation Interface supports double-byte character sets (DBCS) by means of three kinds of character-encoding schemes:

SBCS only Single-byte code pages; for example, US-English.

Both ASCII and EBCDIC SBCS code pages have similar representations.

DBCS only Double-byte code pages; for example, Kanji.

Both ASCII and EBCDIC DBCS code pages have similar representations.

MIXED Code pages that incorporate a combination of single-byte and double-byte characters.

The internal representations of EBCDIC MIXED and ASCII MIXED code pages differ:

- ASCII MIXED: the encoding scheme allows single-byte characters to be distinguished from double-byte characters algorithmically. With this scheme the number of characters entered or displayed is the same as the number of characters in a field.
- EBCDIC MIXED: the encoding scheme requires that control characters within the string switch from single to double byte encoding (and from double to single byte encoding). These control characters are the shift-out (SO) and shift-in (SI) characters.

 With this encoding scheme there may be many more characters in the input or data field than characters displayed or printed.

All MIXED strings are displayed without a space between sequences of single-byte and double-byte characters (unless spaces are explicitly included in these positions within the string).

For graphics, selection of a local identifier (lcid) identifies the code page in force, and therefore whether subsequent character strings are to be interpreted as SBCS, DBCS, ASCII MIXED, or EBCDIC MIXED.

Appendix A. Error Explanations

Recorded Error	Explanation
PMERR_ALREADY_IN_AREA	An attempt was made to begin a new area while an existing area bracket was already open.
PMERR_ALREADY_IN_ELEMENT	An attempt was made to begin a new element while an existing element bracket was already open.
PMERR_ALREADY_IN_PATH	An attempt was made to begin a new path while an existing path bracket was already open.
PMERR_ALREADY_IN_SEG	An attempt was made to open a new segment while an existing segment bracket was already open.
PMERR_APPL_STRUCTURE_TOO_SMALL	The application buffer length, as specified by **InLen** on WinCreateDataStructure or WinModifyDataStructure, or **OutLen** on WinQueryDataStructure, is less than the total length required for the (application) component types.
PMERR_AREA_INCOMPLETE	Either: • A segment has been opened, closed, or drawn. • GpiAssociate was issued while an area bracket was open. • A drawn segment has opened an area bracket and ended without closing it.
PMERR_ARRAY_TOO_LARGE	WinSetValue attempts to insert more than 4 bytes, or WinQueryValue attempts to extract more than 4 bytes.
PMERR_ATOM_NAME_NOT_FOUND	The specified atom name is not in the atom table.
PMERR_BASE_ERROR	An OS/2 base error has occurred. The base error code can be accessed using the OffBinaryData field of the ERRINFO structure returned by WinGetErrorInfo.
PMERR_BITMAP_IN_USE	An attempt was made either to set a bit map into a DC using GpiSetBitmap while it was already selected into an existing DC, or to tag a bit map with a local pattern set identifier (setid) using GpiSetBitmapId while it was already tagged with an existing setid.
PMERR_BITMAP_IS_SELECTED	An attempt was made to delete a bit map while it was selected into a DC.
PMERR_BITMAP_NOT_FOUND	A attempt was made to perform a bit-map operation on a bit map that did not exist.
PMERR_BITMAP_NOT_SELECTED	A attempt was made to perform an operation on presentation space associated with a memory DC that had no selected bit map.
PMERR_BOUNDS_OVERFLOW	An internal overflow error occurred during boundary data accumulation. This can occur if coordinates or matrix transformation elements (or both) are invalid or too large.
PMERR_BUFFER_TOO_SMALL	The supplied buffer was not large enough for the data to be returned.

PMERR_C_LENGTH_TOO_SMALL	The maximum length of the C structure, as specified by **MaxLen** on WinCreateDataStructure, or inherited therefrom by WinModifyDataStructure or WinQueryDataStructure, is less than the total length required for the (C) component types.
PMERR_CALLED_SEG_IS_CHAINED	An attempt was made to call a segment that has a chained attribute set.
PMERR_CAN_NOT_CALL_SPOOLER	An error occurred attempting to call the spooler validation routine. This error is not raised if the spooler is not installed.
PMERR_CANNOT_DEL_PRINTER_DD_REF	Presentation Manager device driver deletion not possible due to a reference.
PMERR_CANNOT_DEL_PRN_ADDR_REF	Printer port deletion not possible due to a reference.
PMERR_CANNOT_DEL_PRN_NAME_REF	Printer deletion not possible due to a reference.
PMERR_CANNOT_DEL_QNAME_REF	Spooler queue deletion not possible due to a reference.
PMERR_CANNOT_DEL_QP_REF	Spooler queue processor deletion not possible due to a reference.
PMERR_CALLED_SEG_IS_CURRENT	An attempt was made to call a segment that is currently open.
PMERR_CALLED_SEG_NOT_FOUND	An attempt was made to call a segment that did not exist.
PMERR_COL_TABLE_NOT_REALIZABLE	An attempt was made to realize a color table that is not realizable.
PMERR_COL_TABLE_NOT_REALIZED	An attempt was made to realize a color table on a device driver that does not support this function.
PMERR_COORDINATE_OVERFLOW	An internal coordinate overflow error occurred. This can occur if coordinates or matrix transformation elements (or both) are invalid or too large.
PMERR_DATA_TOO_LONG	An attempt was made to transfer more than the maximum permitted amount of data (64 512 bytes) using GpiPutData, GpiGetData, or GpiElement.
PMERR_DATATYPE_ENTRY_INVALID	WinSetValue or WinQueryValue has an array element with an invalid data type.
PMERR_DC_IS_ASSOCIATED	An attempt was made to associate a PS with a DC that was already associated or to destroy a DC that was associated.
PMERR_DEL_NOT_ALLOWED	Deletion not possible.
PMERR_DESC_STRING_TRUNCATED	An attempt was made to supply a description string with GpiBeginElement that was greater then the permitted maximum length (251 characters). The string was truncated.
PMERR_DEVICE_DRIVER_ERROR_1	Miscellaneous error available for use by user written device drivers.
PMERR_DEVICE_DRIVER_ERROR_2	Miscellaneous error available for use by user written device drivers.
PMERR_DEVICE_DRIVER_ERROR_3	Miscellaneous error available for use by user written device drivers.
PMERR_DEVICE_DRIVER_ERROR_4	Miscellaneous error available for use by user written device drivers.

PMERR_DEVICE_DRIVER_ERROR_5	Miscellaneous error available for use by user written device drivers.
PMERR_DEVICE_DRIVER_ERROR_6	Miscellaneous error available for use by user written device drivers.
PMERR_DEVICE_DRIVER_ERROR_7	Miscellaneous error available for use by user written device drivers.
PMERR_DEVICE_DRIVER_ERROR_8	Miscellaneous error available for use by user written device drivers.
PMERR_DEVICE_DRIVER_ERROR_9	Miscellaneous error available for use by user written device drivers.
PMERR_DEVICE_DRIVER_ERROR_10	Miscellaneous error available for use by user written device drivers.
PMERR_DOS_ERROR	A DOS call returned an error.
PMERR_DOSOPEN_FAILURE	A DosOpen call made during GpiLoadMetaFile or GpiSaveMetaFile gave a good return code but the file was not opened successfully.
PMERR_DOSREAD_FAILURE	A DosRead call made during GpiLoadMetaFile gave a good return code. However, it failed to read any more bytes although the file length indicated that there were more to be read.
PMERR_DRIVER_NOT_FOUND	The device driver specified with DevPostDeviceModes was not found.
PMERR_DUP_SEG	During GpiPlayMetaFile, while the actual drawing mode was **draw-and-retain** or **retain**, a metafile segment to be stored in the presentation space was found to have the same segment identifier as an existing segment.
PMERR_DUP_SEGNAME	In PicIchg, a called segment has a name that has already been used by another called segment in the input PIF.
PMERR_DUPLICATE_TITLE	The program title specified in the PIBSTRUCT already exists within the same group.
PMERR_DYNAMIC_SEG_SEQ_ERROR	During removal of dynamic segments while processing GpiDrawChain, GpiDrawFrom, or GpiDrawSegment, the internal state indicated that dynamic segment data was still visible after all chained dynamic segments had been processed. This can occur if segments drawn dynamically (including called segments) are modified or removed from the chain while visible.
PMERR_DYNAMIC_SEG_ZERO_INV	An attempt was been made to open a dynamic segment with a segment identifier of zero.
PMERR_ESC_CODE_NOT_SUPPORTED	The code specified with DevEscape is not supported by the target device driver.
PMERR_FONT_AND_MODE_MISMATCH	An attempt was made to draw characters with a character mode and character set that are incompatible. For example, the character specifies an image/raster font when the mode calls for a vector/outline font.
PMERR_FONT_FILE_NOT_LOADED	An attempt was made to unload a font file that was not loaded.
PMERR_FUNCTION_NOT_SUPPORTED	The function is not supported.

PMERR_GREATER_THAN_64K	A data item or array dimension is greater than 65 535.
PMERR_HBITMAP_BUSY	An internal bit map busy error was detected. The bit map was locked by one thread during an attempt to access it from another thread.
PMERR_HDC_BUSY	An internal device context busy error was detected. The device context was locked by one thread during an attempt to access it from another thread.
PMERR_HEAP_MAX_SIZE_REACHED	The heap has reached its maximum size (64KB), and cannot be increased.
PMERR_HEAP_OUT_OF_MEMORY	An attempt to increase the size of the heap failed.
PMERR_HRGN_BUSY	An internal region busy error was detected. The region was locked by one thread during an attempt to access it from another thread.
PMERR_HUGE_FONTS_NOT_SUPPORTED	An attempt was made using GpiSetCharSet, GpiSetPatternSet, GpiSetMarkerSet, or GpiSetAttrs to select a font that is larger than the maximum size (64KB) supported by the target device driver.
PMERR_ID_HAS_NO_BITMAP	No bit map was tagged with the setid specified on a GpiQueryBitmapHandle call.
PMERR_IMAGE_INCOMPLETE	A drawn segment has opened an image bracket and ended without closing it.
PMERR_INCOMPATIBLE_BITMAP	An attempt was made to select a bit map or perform a BitBlt operation on a device context that was incompatible with the format of the bit map.
PMERR_INCOMPATIBLE_METAFILE	An attempt was made to associate a presentation space and a metafile device context with incompatible page units, size or coordinate format; or to play a metafile using the RES_RESET option (to reset the presentation space) to a presentation space that is itself associated with a metafile device context.
PMERR_INCOMPLETE_CONTROL_SEQ	A control data type sequence is incomplete.
PMERR_INCORRECT_DATATYPE	A data type is specified which is incorrect for this function.
PMERR_INCORRECT_DC_TYPE	An attempt was made to perform a bit-map operation on a presentation space associated with a DC of a type that is unable to support bit-map operations.
PMERR_INCORRECT_HSTRUCT	An *HSTRUCT* is non-NULL, and is invalid for one of the following reasons: • It is not the handle of a data structure. • It is the handle of an *ERRINFO* structure which should not be used on this call. • A handle block returned by the bindings to the application has been used for an in-line *HSTRUCT* component.
PMERR_INI_FILE_IS_SYS_OR_USER	User or system initialization file cannot be closed.
PMERR_INSUFF_SPACE_TO_ADD	The initialization file could not be extended to add the required program or group.
PMERR_INSUFFICIENT_DISK_SPACE	The operation terminated through insufficient disk space.

PMERR_INSUFFICIENT_MEMORY	The operation terminated through insufficient memory.
PMERR_INTERNAL_ERROR_n	An internal error has occurred. n is a number that identifies the particular error.
PMERR_INV_ANGLE_PARM	An invalid angle parameter was specified with GpiPartialArc.
PMERR_INV_ARC_CONTROL	An invalid control parameter was specified with GpiFullArc.
PMERR_INV_AREA_CONTROL	An invalid options parameter was specified with GpiBeginArea.
PMERR_INV_ATTR_MODE	An invalid mode parameter was specified with GpiSetAttrMode.
PMERR_INV_BACKGROUND_COL_ATTR	An invalid background color attribute value was specified or the default value was explicitly specified with GpiSetAttrs instead of using the defaults mask.
PMERR_INV_BACKGROUND_MIX_ATTR	An invalid background mix attribute value was specified or the default value was explicitly specified with GpiSetAttrs instead of using the defaults mask.
PMERR_INV_BITBLT_MIX	An invalid **rop** parameter was specified with a GpiBitBlt or GpiWCBitBlt function.
PMERR_INV_BITBLT_STYLE	An invalid options parameter was specified with a GpiBitBlt or GpiWCBitBlt function.
PMERR_INV_BITMAP_DIMENSION	An invalid dimension was specified with a load bit map function.
PMERR_INV_BOX_CONTROL	An invalid control parameter was specified with GpiBox.
PMERR_INV_BOX_ROUNDING_PARM	An invalid corner rounding control parameter was specified with GpiBox.
PMERR_INV_CHAR_ANGLE_ATTR	The default character angle attribute value was explicitly specified with GpiSetAttrs instead of using the defaults mask.
PMERR_INV_CHAR_DIRECTION_ATTR	An invalid character direction attribute value was specified or the default value was explicitly specified with GpiSetAttrs instead of using the defaults mask.
PMERR_INV_CHAR_MODE_ATTR	An invalid character mode attribute value was specified or the default value was explicitly specified with GpiSetAttrs instead of using the defaults mask.
PMERR_INV_CHAR_POS_OPTIONS	An invalid options parameter was specified with GpiCharStringPos or GpiCharStringPosAt.
PMERR_INV_CHAR_SET_ATTR	An invalid character setid attribute value was specified or the default value was explicitly specified with GpiSetAttrs instead of using the defaults mask.
PMERR_INV_CHAR_SHEAR_ATTR	An invalid character shear attribute value was specified or the default value was explicitly specified with GpiSetAttrs instead of using the defaults mask.
PMERR_INV_CLIP_PATH_OPTIONS	An invalid options parameter was specified with GpiSetClipPath.

PMERR_INV_CODEPAGE	An invalid code-page parameter was specified with GpiSetCp.
PMERR_INV_COLOR_ATTR	An invalid color attribute value was specified or the default value was explicitly specified with GpiSetAttrs instead of using the defaults mask.
PMERR_INV_COLOR_DATA	Invalid color table definition data was specified with GpiCreateLogColorTable.
PMERR_INV_COLOR_FORMAT	An invalid format parameter was specified with GpiCreateLogColorTable.
PMERR_INV_COLOR_INDEX	An invalid color index parameter was specified with GpiQueryRGBColor.
PMERR_INV_COLOR_OPTIONS	An invalid options parameter was specified with a logical color table or color query function.
PMERR_INV_COLOR_START_INDEX	An invalid starting index parameter was specified with a logical color table or color query function.
PMERR_INV_CONV	Invalid conversion-type parameter in PicIchg.
PMERR_INV_COORD_OFFSET	An invalid coordinate offset value was specified.
PMERR_INV_COORD_SPACE	An invalid src or targ coordinate space parameter was specified with GpiConvert.
PMERR_INV_COORDINATE	An invalid coordinate value was specified.
PMERR_INV_CORRELATE_DEPTH	An invalid maxdepth parameter was specified with GpiCorrelateSegment, GpiCorrelateFrom, or GpiCorrelateChain.
PMERR_INV_CORRELATE_TYPE	An invalid type parameter was specified with GpiCorrelateSegment, GpiCorrelateFrom, or GpiCorrelateChain.
PMERR_INV_CURSOR_BITMAP	An invalid pointer was referenced with WinSetPointer.
PMERR_INV_DC_DATA	An invalid data parameter was specified with DevOpenDC.
PMERR_INV_DC_TYPE	An invalid type parameter was specified with DevOpenDC, or a function was issued that is invalid for a OD_METAFILE_NOQUERY device context.
PMERR_INV_DEV_MODES_OPTIONS	An invalid options parameter was specified with DevPostDeviceModes.
PMERR_INV_DEVICE_NAME	An invalid devicename parameter was specified with DevPostDeviceModes.
PMERR_INV_DRAW_BORDER_OPTION	An invalid **option** parameter was specified with WinDrawBorder.
PMERR_INV_DRAW_CONTROL	An invalid **control** parameter was specified with GpiSetDrawControl or GpiQueryDrawControl.
PMERR_INV_DRAW_VALUE	An invalid value parameter was specified with GpiSetDrawControl.
PMERR_INV_DRAWING_MODE	An invalid mode parameter was specified with GpiSetDrawControl not **draw-and-retain** or **draw**.
PMERR_INV_DRIVER_NAME	A driver name was specified which has not been installed.
PMERR_INV_EDIT_MODE	An invalid mode parameter was specified with GpiSetEditMode.

PMERR_INV_ELEMENT_OFFSET	An invalid off (offset) parameter was specified with GpiQueryElement.
PMERR_INV_ELEMENT_POINTER	An attempt was made to issue GpiPutData with the element pointer not pointing at the last element.
PMERR_INV_ESC_CODE	An invalid code parameter was specified with DevEscape.
PMERR_INV_ESCAPE_DATA	An invalid data parameter was specified with DevEscape.
PMERR_INV_FILL_PATH_OPTIONS	An invalid options parameter was specified with GpiFillPath.
PMERR_INV_FIRST_CHAR	An invalid firstchar parameter was specified with GpiQueryWidthTable.
PMERR_INV_FONT_ATTRS	An invalid attrs parameter was specified with GpiCreateLogFont.
PMERR_INV_FONT_FILE_DATA	The font file specified with GpiLoadFonts or GpiQueryFontFileDescriptions contains invalid data.
PMERR_INV_FOR_THIS_DC_TYPE	An attempt has been made to issue GpiRemoveDynamics or GpiDrawDynamics to a presentation space associated with a metafile device context.
PMERR_INV_FORMS_CODE	An invalid forms code parameter was specified with DevQueryHardcopyCaps.
PMERR_INV_GEOM_LINE_WIDTH_ATTR	An invalid geometric line width attribute value was specified.
PMERR_INV_GETDATA_CONTROL	An invalid format parameter was specified with GpiGetData.
PMERR_INV_GRAPHICS_FIELD	An invalid field parameter was specified with GpiSetGraphicsField.
PMERR_INV_HBITMAP	An invalid bit-map handle was specified.
PMERR_INV_HDC	An invalid device-context handle or (micro presentation space) presentation-space handle was specified.
PMERR_INV_HMF	An invalid metafile handle was specified.
PMERR_INV_HPS	An invalid presentation-space handle was specified.
PMERR_INV_HRGN	An invalid region handle was specified.
PMERR_INV_ID	An invalid **PSid** parameter was specified with GpiRestorePS.
PMERR_INV_IMAGE_DATA_LENGTH	An invalid **Length** parameter was specified with GpiImage. There is a mismatch between the image size and the data length.
PMERR_INV_IMAGE_DIMENSION	An invalid **ImageSize** parameter was specified with GpiImage.
PMERR_INV_IMAGE_FORMAT	An invalid **Format** parameter was specified with GpiImage.
PMERR_INV_IN_AREA	An attempt was made to issue a function invalid inside an area bracket. This can be detected while the actual drawing mode is **draw** or **draw-and-retain** or during segment drawing or correlation functions.

PMERR_INV_IN_ELEMENT	An attempt was made to issue a function invalid inside an element bracket.
PMERR_INV_IN_IMAGE	An attempt was made to issue a function invalid inside an element bracket.
PMERR_INV_IN_PATH	An attempt was made to issue a function invalid inside a path bracket.
PMERR_INV_IN_RETAIN_MODE	An attempt was made to issue a function (for example, query) that is invalid when the actual drawing mode is not **draw** or **draw-and-retain.**
PMERR_INV_IN_SEG	An attempt was made to issue a function invalid inside a segment bracket.
PMERR_INV_IN_VECTOR_SYMBOL	An invalid order was detected inside a vector symbol definition while drawing a vector (outline) font.
PMERR_INV_INFO_TABLE	An invalid bit-map info table was specified with a bit-map operation.
PMERR_INV_LENGTH_OR_COUNT	An invalid length or count parameter was specified.
PMERR_INV_LINE_END_ATTR	An invalid line end attribute value was specified.
PMERR_INV_LINE_JOIN_ATTR	An invalid line join attribute value was specified.
PMERR_INV_LINE_TYPE_ATTR	An invalid line type attribute value was specified or the default value was explicitly specified with GpiSetAttrs instead of using the defaults mask.
PMERR_INV_LINE_WIDTH_ATTR	An invalid line width attribute value was specified or the default value was explicitly specified with GpiSetAttrs instead of using the defaults mask.
PMERR_INV_MARKER_BOX_ATTR	An invalid marker box attribute value was specified.
PMERR_INV_MARKER_SET_ATTR	An invalid marker set attribute value was specified or the default value was explicitly specified with GpiSetAttrs instead of using the defaults mask.
PMERR_INV_MARKER_SYMBOL_ATTR	An invalid marker symbol attribute value was specified or the default value was explicitly specified with GpiSetAttrs instead of using the defaults mask.
PMERR_INV_MATRIX_ELEMENT	An invalid transformation matrix element was specified.
PMERR_INV_MAX_HITS	An invalid maxhits parameter was specified with GpiCorrelateSegment, GpiCorrelateFrom, or GpiCorrelateChain.
PMERR_INV_METAFILE	An invalid metafile was specified with GpiPlayMetaFile.
PMERR_INV_METAFILE_LENGTH	An invalid length parameter was specified with GpiSetMetaFileBits or GpiQueryMetaFileBits.
PMERR_INV_METAFILE_OFFSET	An invalid length parameter was specified with GpiSetMetaFileBits or GpiQueryMetaFileBits.
PMERR_INV_MICROPS_DRAW_CONTROL	A draw control parameter was specified with GpiSetDrawControl that is invalid in a micro presentation space.
PMERR_INV_MICROPS_FUNCTION	An attempt was made to issue a function that is invalid in a micro presentation space.

PMERR_INV_MICROPS_ORDER	An attempt was made to play a metafile containing orders that are invalid in a micro presentation space.
PMERR_INV_MIX_ATTR	An invalid mix attribute value was specified or the default value was explicitly specified with GpiSetAttrs instead of using the defaults mask.
PMERR_INV_MODE_FOR_OPEN_DYN	An attempt was made to open a segment with the ATTR_DYNAMIC segment set, while the drawing mode was set to DM_DRAW or DM_DRAWANDRETAIN.
PMERR_INV_MODE_FOR_REOPEN_SEG	An attempt was made to reopen an existing segment while the drawing mode was set to DM_DRAW or DM_DRAWANDRETAIN.
PMERR_INV_MODIFY_PATH_MODE	An invalid mode parameter was specified with GpiModifyPath.
PMERR_INV_MULTIPLIER	An invalid multiplier parameter was specified with GpiPartialArc or GpiFullArc.
PMERR_INV_NESTED_FIGURES	Nested figures have been detected within a path definition.
PMERR_INV_OR_INCOMPAT_OPTIONS	An invalid or incompatible (with micro presentation space) options parameter was specified with GpiCreatePS or GpiSetPS.
PMERR_INV_ORDER_LENGTH	An invalid order length was detected during GpiPutData or segment drawing.
PMERR_INV_ORDERING_PARM	An invalid order parameter was specified with GpiSetSegmentPriority.
PMERR_INV_OUTSIDE_DRAW_MODE	An attempt was made to issue a GpiSavePS or GpiRestorePS function, or an output only function (for example, GpiPaintRegion) from GpiPlayMetaFile without drawing mode not to DM_DRAW.
PMERR_INV_PAGE_VIEWPORT	An invalid viewport parameter was specified with GpiSetPageViewport.
PMERR_INV_PATH_CONVERT_OPTIONS	An invalid options parameter was specified with GpiOutlinePath.
PMERR_INV_PATH_ID	An invalid path identifier parameter was specified.
PMERR_INV_PATTERN_ATTR	An invalid pattern symbol attribute value was specified or the default value was explicitly specified with GpiSetAttrs instead of using the defaults mask.
PMERR_INV_PATTERN_REF_PT_ATTR	An invalid refpoint attribute value was specified.
PMERR_INV_PATTERN_SET_ATTR	An invalid pattern set attribute value was specified or the default value was explicitly specified with GpiSetAttrs instead of using the defaults mask.
PMERR_INV_PATTERN_SET_FONT	An attempt was made to use an unsuitable font as a pattern set.
PMERR_INV_PICK_APERTURE_OPTION	An invalid options parameter was specified with GpiSetPickApertureSize.
PMERR_INV_PICK_APERTURE_POSN	An invalid pick aperture position was specified.
PMERR_INV_PICK_APERTURE_SIZE	An invalid size parameter was specified with GpiSetPickApertureSize.

PMERR_INV_PLAY_METAFILE_OPTION	An invalid option parameter was specified with GpiPlayMetaFile.
PMERR_INV_PRIMITIVE_TYPE	An invalid primitive type parameter was specified with GpiSetAttrs or GpiQueryAttrs.
PMERR_INV_PS_SIZE	An invalid size parameter was specified with GpiCreatePS or GpiSetPS.
PMERR_INV_PUTDATA_FORMAT	An invalid format parameter was specified with GpiPutData.
PMERR_INV_QUERY_ELEMENT_NO	An invalid start parameter was specified with DevQueryCaps.
PMERR_INV_RECT	An invalid rectangle parameter was specified.
PMERR_INV_REGION_CONTROL	An invalid control parameter was specified with GpiQueryRegionRects.
PMERR_INV_REGION_MIX_MODE	An invalid mode parameter was specified with GpiCombineRegion.
PMERR_INV_REPLACE_MODE_FUNC	An attempt was made to issue GpiPutData with the editing mode set to SEGEM_REPLACE.
PMERR_INV_RESERVED_FIELD	An invalid reserved field was specified.
PMERR_INV_RESET_OPTIONS	An invalid options parameter was specified with GpiResetPS.
PMERR_INV_RGBCOLOR	An invalid rgb color parameter was specified with GpiQueryNearestColor or GpiQueryColor.
PMERR_INV_SCAN_START	An invalid scanstart parameter was specified with GpiQueryNearestColor or GpiQueryColorIndex.
PMERR_INV_SEG_ATTR	An invalid attribute parameter was specified with GpiSetSegmentAttrs, GpiQuerySegmentAttrs, GpiSetInitialSegmentAttrs, or GpiQueryInitialSegmentAttrs.
PMERR_INV_SEG_ATTR_VALUE	An invalid attribute value parameter was specified with GpiSetSegmentAttrs or GpiSetInitialSegmentAttrs.
PMERR_INV_SEG_NAME	An invalid segment identifier was specified.
PMERR_INV_SEGLEN	In PicIchg, an order length exceeds the remaining segment length in the input PIF.
PMERR_INV_SEG_OFFSET	An invalid offset parameter was specified with GpiPutData.
PMERR_INV_SETID	An invalid setid parameter was specified.
PMERR_INV_SHARPNESS_PARM	An invalid sharpness parameter was specified with GpiPolyFilletSharp.
PMERR_INV_STOP_DRAW_VALUE	An invalid value parameter was specified with GpiSetStopDraw.
PMERR_INV_TRANSFORM_TYPE	An invalid options parameter was specified with a transform matrix function.
PMERR_INV_TYPE	Invalid file-type parameter in PicPrint.
PMERR_INV_USAGE_PARM	An invalid options parameter was specified with GpiCreateBitmap.
PMERR_INV_VIEWING_LIMITS	An invalid limits parameter was specified with GpiSetViewingLimits.
PMERR_INV_VIEWLIM	In PicIchg, a set viewing limits order has an inconsistent mask and order length in the input PIF.

PMERR_INV_XFORM	In PicIchg, a set (default) viewing transform order has an in consistent mask and order length in the input PIF.
PMERR_INV_3DCOORD	In PicIchg, an order specifying 3-dimensional coordinates has been found in the input PIF.
PMERR_INVALID_ARRAY_COUNT	WinSetValue or WinQueryValue has an invalid count, that is, less than or equal to zero.
PMERR_INVALID_ARRAY_SIZE	A control data type array size is invalid.
PMERR_INVALID_ASCIIZ	The profile string is not a valid zero-terminated string.
PMERR_INVALID_ATOM	The specified atom does not exist in the atom table.
PMERR_INVALID_ATOM_NAME	An invalid atom name string was passed.
PMERR_INVALID_BUNDLE_TYPE	An invalid bundle type was passed.
PMERR_INVALID_CHARACTER_INDEX	On WinNextChar or WinPrevChar, a character index is invalid, that is, it is less than 1 or is greater than the string length+1.
PMERR_INVALID_CONTROL_DATATYPE	An invalid control data type was specified.
PMERR_INVALID_CONTROL_SEQ_INDEX	There is an invalid index in a control data type sequence (for array, length, offset or *MPARAM*) that is, the index is to a non-existent or non-numeric entry.
PMERR_INVALID_DATATYPE	An invalid data type was specified.
PMERR_INVALID_DST_CODEPAGE	An invalid destination code page was passed with WinCpTranslateChar or WinCpTranslateString.
PMERR_INVALID_FLAG	An invalid bit was set for a parameter. Use constants defined by Presentation Manager for options, and do not set any reserved bits.
PMERR_INVALID_ERRORINFO_HANDLE	On WinFreeErrorInfo, the *HSTRUCT* is not the handle of an *ERRINFO* structure, that is, it was not created by WinGetErrorInfo.
PMERR_INVALID_FREE_MESSAGE_ID	An invalid message identifier was specified on WinFreeMsg. The call has completed by assuming the message parameter and reply data types to be *BIT32*.
PMERR_INVALID_GROUP_HANDLE	An invalid program-group handle was specified.
PMERR_INVALID_HACCEL	An invalid accelerator-table handle was specified.
PMERR_INVALID_HATOMTBL	An invalid atom-table handle was specified.
PMERR_INVALID_HEAP_POINTER	An invalid pointer was found within the heap.
PMERR_INVALID_HEAP_SIZE_PARM	Invalid data was found within the heap.
PMERR_INVALID_HEAP_SIZE_WORD	Invalid data was found within the heap.
PMERR_INVALID_HENUM	An invalid enumeration handle was specified.
PMERR_INVALID_HHEAP	An invalid heap handle was specified.
PMERR_INVALID_HMQ	An invalid message-queue handle was specified.
PMERR_INVALID_HPTR	An invalid pointer handle was specified.
PMERR_INVALID_HSTRUCT	An invalid (null) structure handle was specified.
PMERR_INVALID_HWND	An invalid window handle was specified.
PMERR_INVALID_INI_FILE_HANDLE	An invalid initialization file handle was specified.
PMERR_INVALID_INTEGER_ATOM	The specified atom is not a valid integer atom.

PMERR_INVALID_MESSAGE_ID	A message identifier is invalid.
PMERR_INVALID_NUMBER_OF_PARMS	The number of parameters is invalid.
PMERR_INVALID_NUMBER_OF_TYPES	WinCreateDataStructure has an invalid number (zero) of types.
PMERR_INVALID_PARAMETER	An application parameter value is invalid for its converted PM type. For example: a 4-byte value outside the range –32 768 to +32 767 cannot be converted to a SHORT, and a negative number cannot be converted to a ULONG or USHORT.
PMERR_INVALID_PARAMETER_TYPE	A parameter type is invalid for a bundle mask.
PMERR_INVALID_PARM	A parameter to the function contained invalid data.
PMERR_INVALID_PROGRAM_HANDLE	An invalid program handle was specified.
PMERR_INVALID_SELECTOR	An invalid segment selector was specified with WinCreateHeap.
PMERR_INVALID_SESSION_ID	The specified session identifier is invalid. Either zero (for the application's own session) or a valid identifier must be specified.
PMERR_INVALID_SRC_CODEPAGE	An invalid source code page was passed with WinCpTranslateChar or WinCpTranslateString.
PMERR_INVALID_STRING_PARM	The specified string parameter is invalid.
PMERR_INVALID_SWITCH_HANDLE	An invalid task-list entry handle was specified.
PMERR_INVALID_TARGET_HANDLE	An invalid target program-group handle was specified.
PMERR_INVALID_TITLE	The specified program or group title is too long or contains invalid characters. See WinCreateGroup for the syntax rules.
PMERR_INVALID_TYPE_FOR_LENGTH	The data type for a control length is invalid.
PMERR_INVALID_TYPE_FOR_MPARAM	The message parameter type for a control *MPARAM* is invalid, that is, not mparam1, mparam2 or mreply.
PMERR_INVALID_TYPE_FOR_OFFSET	The data type for a control offset is invalid.
PMERR_INVALID_WINDOW	The window specified with a task list call is not a valid frame window.
PMERR_KERNING_NOT_SUPPORTED	Kerning was requested on GpiCreateLogFont call to a presentation space associated with a device context that does not support kerning.
PMERR_LABEL_NOT_FOUND	The specified element label did not exist.
PMERR_MATRIX_OVERFLOW	An internal overflow error occurred during matrix multiplication. This can occur if coordinates or matrix transformation elements (or both) are invalid or too large.
PMERR_MEMORY_ALLOC	An error occurred during memory management.
PMERR_MEMORY_ALLOCATION_ERR	An error occurred during memory management.
PMERR_MEMORY_DEALLOCATION_ERR	An error occurred during memory management.
PMERR_METAFILE_INTERNAL_ERROR	An internal inconsistency has been detected during metafile unlock processing.
PMERR_METAFILE_IN_USE	An attempt has been made to access a metafile that is in use by another thread.

PMERR_METAFILE_LIMIT_EXCEEDED	The maximum permitted metafile size limit was exceeded during metafile recording.
PMERR_NEGATIVE_STRCOND_DIM	A negative array dimension was passed for a *STRCOND* data type.
PMERR_NO_BITMAP_SELECTED	An attempt has been made to operate on a memory device context that has no bit map selected.
PMERR_NO_CURRENT_ELEMENT	An attempt has been made to issue GpiQueryElementType or GpiQueryElement while there is no currently open element.
PMERR_NO_CURRENT_SEG	An attempt has been made to issue GpiQueryElementType or GpiQueryElement while there is no currently open segment.
PMERR_NO_METAFILE_RECORD_HANDLE	The metafile record handle was not found during metafile recording, or DevEscape (DEVESC_STARTDOC) was not issued when drawing to a OD_QUEUED device context with a **datatype** field of PM_Q_STD.
PMERR_NO_SPACE	The limit on the number of task list entries has been reached with WinAddSwitchEntry.
PMERR_NOT_CREATED_BY_DEVOPENDC	An attempt has been made to destroy a device context using DevCloseDC that was not created using DevOpenDC.
PMERR_NOT_CURRENT_PL_VERSION	An unexpected data format was found in the initialization file.
PMERR_NOT_IN_AREA	An attempt was made to end an area using GpiEndArea or during segment drawing while not in an area bracket.
PMERR_NOT_IN_DRAW_MODE	An attempt was made to issue GpiSavePS or GpiRestorePS while the drawing mode was not set to DM_DRAW.
PMERR_NOT_IN_ELEMENT	An attempt was made to end an element using GpiEndElement or during segment drawing while not in an element bracket.
PMERR_NOT_IN_IDX	The application name, key-name or program handle was not found.
PMERR_NOT_IN_IMAGE	An attempt was made to end an image during segment drawing while not in an image bracket.
PMERR_NOT_IN_PATH	An attempt was made to end a path using GpiEndPath or during segment drawing while not in an path bracket.
PMERR_NOT_IN_RETAIN_MODE	An attempt was made to issue a segment editing element function that is invalid when the actual drawing mode is not set to **retain**.
PMERR_NOT_IN_SEG	An attempt was made to end a segment using GpiCloseSegment while not in a segment bracket.
PMERR_NOT_SELF_DESCRIBING_DTYP	A data type is not self-describing.
PMERR_OPENING_INI_FILE	Unable to open initialization file (due to lack of disk space for example).
PMERR_ORDER_TOO_BIG	An internal size limit was exceeded while converting orders from short to long format during GpiPutData processing. An order was too long to convert.

PMERR_PARAMETER_OUT_OF_RANGE	The value of a parameter was not within the defined valid range for that parameter.
PMERR_PATH_INCOMPLETE	An attempt was made to open or close a segment either directly or during segment drawing, or to issue GpiAssociate while there is an open path bracket.
PMERR_PATH_LIMIT_EXCEEDED	An internal size limit was exceeded during path or area processing.
PMERR_PATH_UNKNOWN	An attempt was made to perform a path function on a path that did not exist.
PMERR_PEL_IS_CLIPPED	An attempt was made to query a pel that had been clipped using GpiQueryPel.
PMERR_PEL_NOT_AVAILABLE	An attempt was made to query a pel that did not exist in GpiQueryPel (for example, a memory device context with no selected bit map).
PMERR_PROLOG_ERROR	A prolog error was detected during drawing. Segment prologs are used internally within retained segments and also appear in metafiles. This error can also arise from an End Prolog order that is outside a prolog.
PMERR_PRINTER_DD_NOT_DEFINED	The Presentation Manager device driver has not been defined.
PMERR_PRINTER_QUEUE_NOT_DEFINED	The spooler queue for the printer has not been defined.
PMERR_PRN_ADDR_IN_USE	A printer is already defined on the port.
PMERR_PRN_ADDR_NOT_DEFINED	The printer port has not been defined.
PMERR_PRN_NAME_NOT_DEFINED	The printer has not been defined.
PMERR_PS_BUSY	An attempt was made to access the presentation space from more than one thread simultaneously.
PMERR_PS_IS_ASSOCIATED	An attempt was made to destroy a presentation or associate a presentation space that is still associated with a device context.
PMERR_REALIZE_NOT_SUPPORTED	An attempt was made to create a realizable logical color table on a device driver that does not support this function.
PMERR_REGION_IS_CLIP_REGION	An attempt was made to perform a region operation on a region that is selected as a clip region.
PMERR_RESOURCE_DEPLETION	An internal resource depletion error has occurred.
PMERR_RESOURCE_NOT_FOUND	The specified resource identity could not be found.
PMERR_SEG_AND_REFSEG_ARE_SAME	The segid and refsegid specified with GpiSetSegmentPriority were the same.
PMERR_SEG_CALL_STACK_EMPTY	A call stack empty condition was detected when attempting a pop function during GpiPop or segment drawing.
PMERR_SEG_CALL_STACK_FULL	A call stack full condition was detected when attempting to call a segment using GpiCallSegmentMatrix, attempting to preserve an attribute, or during segment drawing.
PMERR_SEG_IS_CURRENT	An attempt was made to issue GpiGetData to a segment that was currently open.

PMERR_SEG_NOT_CHAINED	An attempt was made to issue GpiDrawFrom, GpiCorrelateFrom or GpiQuerySegmentPriority for a segment that was not chained.
PMERR_SEG_NOT_FOUND	A segment identifier was specified that did not exist.
PMERR_SEG_OVFLOW	In PicIchg, the input PIF has more than 1000 called segments. This has overflowed an internal buffer in PicIchg.
PMERR_SEG_STORE_LIMIT_EXCEEDED	The maximum permitted retained segment store size limit was exceeded.
PMERR_SETID_IN_USE	An attempt was made to specify a setid that was already in use as the currently selected character, marker or pattern set.
PMERR_SETID_NOT_FOUND	An attempt was made to delete a setid that did not exist.
PMERR_SMB_OVFLOW	In PicIchg, the input PIF has more than 100 symbol sets defined. This has overflowed an internal buffer in PicIchg.
PMERR_SPL_CANNOT_OPEN_FILE	Unable to open the file.
PMERR_SPL_DD_NOT_FOUND	The Presentation Manager device driver definition could not be found.
PMERR_SPL_DEVICE_ALREADY_EXISTS	The device already exists.
PMERR_SPL_DEVICE_LIMIT_REACHED	The limit on the number of devices has been reached.
PMERR_SPL_DEVICE_NOT_INSTALLED	The device has not been installed.
PMERR_SPL_DRIVER_ERROR	No Presentation Manager device driver supplied or found.
PMERR_SPL_DRIVER_NOT_INSTALLED	The Presentation Manager device driver has not been installed.
PMERR_SPL_FILE_NOT_FOUND	Unable to find the file.
PMERR_SPL_HARD_NETWORK_ERROR	Hard network error.
PMERR_SPL_INI_FILE_ERROR	Error accessing the initialization file.
PMERR_SPL_INV_DATATYPE	The spool file data type is invalid.
PMERR_SPL_INV_DRIVER_DATATYPE	The data type is invalid for the Presentation Manager device driver.
PMERR_SPL_INV_FORMS_CODE	The forms code for the job is invalid.
PMERR_SPL_INV_HSPL	The spooler handle is invalid.
PMERR_SPL_INV_JOB_ID	The job id is invalid.
PMERR_SPL_INV_LENGTH_OR_COUNT	The length or count is invalid.
PMERR_SPL_INV_PRIORITY	The priority for the job is invalid.
PMERR_SPL_INV_PROCESSOR_DATTYPE	The data type is invalid for the spooler queue processor.
PMERR_SPL_INV_QUEUE_NAME	The spooler queue name is invalid.
PMERR_SPL_INV_TOKEN	The token is invalid.
PMERR_SPL_JOB_NOT_PRINTING	The print job is not printing.
PMERR_SPL_JOB_PRINTING	The print job is already printing.
PMERR_SPL_MANY_QUEUES_ASSOC	More than one queue has been associated with the printer.

PMERR_SPL_NO_CURRENT_FORMS_CODE	There is no current forms code defined to the Presentation Manager device driver.
PMERR_SPL_NO_DATA	No data supplied or found.
PMERR_SPL_NO_DEFAULT_QUEUE	There is no default spooler queue for the printer.
PMERR_SPL_NO_DISK_SPACE	There is not enough free disk space.
PMERR_SPL_NO_FREE_JOB_ID	There is no free job id available.
PMERR_SPL_NO_MEMORY	There is not enough free memory.
PMERR_SPL_NO_QUEUES_ASSOCIATED	A queue has not been associated with the printer.
PMERR_SPL_NO_SUCH_LOG_ADDRESS	The logical address does not exist (that is, it is not defined in the initialization file).
PMERR_SPL_NOT_AUTHORISED	Not authorized to perform the operation.
PMERR_SPL_PRINT_ABORT	The job has already been aborted.
PMERR_SPL_PRINTER_NOT_FOUND	The printer definition could not be found.
PMERR_SPL_PROCESSOR_ERROR	No spooler queue processor supplied or found.
PMERR_SPL_PROCESSOR_NOT_INST	The spooler queue processor has not been installed.
PMERR_SPL_QUEUE_ALREADY_EXISTS	The spooler queue already exists.
PMERR_SPL_QUEUE_ERROR	No spooler queue supplied or found.
PMERR_SPL_QUEUE_NOT_EMPTY	The spooler queue contains print jobs.
PMERR_SPL_QUEUE_NOT_FOUND	The spooler queue definition could not be found.
PMERR_SPL_SPOOLER_NOT_INSTALLED	The spooler is not installed.
PMERR_SPL_STATUS_STRING_TRUNC	The print job status string has been truncated.
PMERR_SPL_TEMP_NETWORK_ERROR	Temporary network error.
PMERR_SPL_TOO_MANY_OPEN_FILES	Too many open files.
PMERR_SPOOLER_QP_NOT_DEFINED	The spooler queue processor has not been defined.
PMERR_STOP_DRAW_OCCURRED	Segment drawing or GpiPlayMetaFile was stopped prematurely in response to a GpiSetStopDraw request.
PMERR_TOO_MANY_METAFILES_IN_USE	The maximum number of metafiles allowed for a given process was exceeded.
PMERR_TRUNCATED_ORDER	An incomplete order was detected during segment processing.
PMERR_UNABLE_TO_CLOSE_DEVICE	Unable to close the print device (for example, powered off or offline).
PMERR_UNCHAINED_SEG_ZERO_INV	An attempt was made to open segment with segment identifier zero and the ATTR_CHAINED segment attribute not specified.
PMERR_UNKNOWN_BUNDLE_TYPE	Unknown bundle-type primitive.
PMERR_UNSUPPORTED_ATTR	An unsupported attribute was specified in the attrmask with GpiSetAttrs or GpiQueryAttrs.
PMERR_UNSUPPORTED_ATTR_VALUE	An attribute value was specified with GpiSetAttrs that is not supported.
PMERR_WINDOW_LOCK_OVERFLOW	An overflow occurred for the use count of a window with WinLockWindow.

PMERR_WINDOW_LOCK_UNDERFLOW	An attempt was mode to decrement the use count of a window below zero with WinLockWindow.
PMERR_WINDOW_NOT_LOCKED	The window specified in WinSendMsg was not locked.

Appendix B. Standard Bit-Map Formats

There are four standard bit-map formats. All device drivers have to be able to translate between any of these formats and their own internal formats. The standard formats are:

Bitcount	Planes
1	1
4	1
8	1
24	1

These formats are chosen because they are identical or similar to all formats commonly used by raster devices. Only single-plane formats are standard, but it is very easy to convert these to any multiple-plane format used internally by a device.

Bit-Map Data

The pel data is stored in the bit map in the order that the coordinates appear on a display screen. That is, the pel in the lower-left corner is the first in the bit map. Pels are scanned to the right, and upward, from that position. The bits of the first pel are stored, beginning with the most significant bits of the first byte. The data for pels in each scan line is packed together tightly, but all scan lines are padded at the end, so that each one begins on a ULONG boundary.

Bit-Map Information Tables

Each standard-format bit map must be accompanied by a bit-map information table. Because the standard-format bit maps are intended to be traded between devices, the color indexes in the bit map are meaningless without more information; for a description of this structure, see BITMAPINFO on page 2-2.

Some calls use a structure that is similar to *BITMAPINFO*, but does not have the **color** array; for a description of this structure, see BITMAPINFOHEADER on page 2-3.

Bit-Map Example

To make the ordering of all the bytes clear, consider this simple example of a 5-by-3 array of colored pels:

```
Red   Green Blue  Red   Green
Blue  Red   Green Blue  Red
Green Blue  Red   Green Blue

ULONG ExampleBitmap[]  {
    0x23,0x12,0x30,0x00             /* bottom line */
    0x31,0x23,0x10,0x00             /* middle line */
    0x12,0x31,0x20,0x00             /* top line    */
};

#define BLACK  0x00000000L
#define RED    0x00FF0000L
#define GREEN  0x0000FF00L
#define BLUE   0x000000FFL

struct BitmapInfoTable ExampleInfo = {
       5,                           /* width       */
       3,                           /* height      */
       1,                           /* planes      */
       4,                           /* bitcount    */
       BLACK,RED,GREEN,BLUE,        /* color table */
       BLACK,BLACK,BLACK,BLACK,
       BLACK,BLACK,BLACK,BLACK,
       BLACK,BLACK,BLACK,BLACK
};
```

Bit-Map File Format

Presentation Manager uses the same file format for bit maps, icons, and pointers in resource files. In the following description, "bit map" refers to bit maps, icons, and pointers unless otherwise specified.

The file format consists of two sections. The first section contains general information about the type, dimensions, and other attributes of the resource. The second section contains data describing the pels that make up the bit map(s), and is in the format specified in "Bit-Map Data" on page B-1.

The first section consists of one or two BITMAPFILEHEADER structures, which have the following format:

```
typedef struct _BITMAPFILEHEADER { /* bfh */
    USHORT            usType;
    ULONG             cbSize;
    INT               xHotspot;
    INT               yHotspot;
    ULONG             offBits;
    BITMAPINFOHEADER  bmp;
} BITMAPFILEHEADER;
typedef BITMAPFILEHEADER FAR *PBITMAPFILEHEADER;
```

BITMAPINFOHEADER is a standard data type (see above, and also BITMAPINFOHEADER on page 2-3).

The fields in BITMAPFILEHEADER have these meanings:

usType Type of resource the file contains. The valid values are:

BFT_BMAP (X'4D42' — 'BM' for bit maps)
BFT_ICON (X'4349' — 'IC' for icons)
BFT_POINTER (X'5450' — 'PT' for pointers)
BFT_COLORICON (X'4943' — 'CI' for color icons)
BFT_COLORPOINTER (X'5043' — 'CP' for color pointers).

cbSize Size of the file in bytes.

xHotspot, yHotspot Coordinates of the hotspot for icons and pointers. This field is ignored for bit maps.

offBits Offset in bytes to the beginning of the bit-map pel data in the file.

For icons and pointers, the **bitmapheight** field in **bmp** is actually twice the pel height of the image that appears on the screen. This is because these types actually contain two full bit-map pel definitions. The first bit-map definition is the XOR mask, which contains invert information (0 = no invert, 1 = invert) for the pointer or icon. The second is the AND mask, which determines whether the pointer or the screen is shown (0 = black/white, 1 = screen/inverse screen).

For color icons or pointers, there are two bit-maps involved: one that is black and white and consists of an AND and an XOR mask, and one that is color that defines the color content.

The **bitmapheight** field in the *BITMAPINFOHEADER* structure for the color bit-map must be the real height, that is, half the value specified for the black and white bit-map. The **bitmapwidth** fields must be the same.

The following table shows how these two bit-maps are used for a color icon or pointer:

XOR	AND	COLOR	
1	1	x	Invert screen
0	0	x	Use color x
0	1	x	Transparency
1	0	x	Use color x

For color icons or pointers, two BITMAPFILEHEADER structures are required:

```
BITMAPFILEHEADER     with usType BFT_COLORICON or BFT_COLORPOINTER
BITMAPFILEHEADER     with same usType
..
bits for one bit-map
..
..
bits for other bit-map
..
```

The usType for the first BITMAPFILEHEADER is either BFT_COLORICON or BFT_COLORPOINTER. This means that a second BITMAPFILEHEADER is present as part of the definition of a color icon or pointer. The first BITMAPFILEHEADER structure contains the information for the black and white AND and XOR masks, while the second BITMAPFILEHEADER structure contains the information for the color part of the pointer or icon.

Appendix C. The Font-File Format

The OS/2 Version 1.2 font-file format consists of two sections. The first section contains the general attributes of the font, and describes features such as its typeface, style, and nominal size. The second section contains the actual definitions of the characters belonging to the font.

The font resource is a set of self-defining records of the form:

```
typedef struct _RECORD {
        ULONG   ulIdentity;     /* structure identity code */
        ULONG   ulSize;         /* structure size in bytes */
         .                      /* data */
         .
         .
        } RECORD;
```

A font starts with a special font-signature structure and ends with an ending structure. The font signature has the form:

```
typedef struct _FONTSIGNATURE {
        ULONG   ulIdentity;
        ULONG   ulSize;
        CHAR    achSignature [12]
        } FONTSIGNATURE;
```

where:

```
ulIdentity   = X'FFFFFFFE'
ulSize       = 20
achSignature = "OS/2 FONT   "
```

The font end structure has the form:

```
typedef struct _ENDFONT{
        ULONG   ulIdentity;
        ULONG   ulSize;
        }ENDFONT
```

where:

```
ulIdentity   = X'FFFFFFFE'
ulSize       = 8
```

All records should be in the order of their identity fields.

There are three records in a font resource between the font signature and the font end:

- The font metrics
- The font character definitions
- The pair kerning table.

Following compilation, the records in the resource are in the order defined above.

Font Metrics

The following structure contains the physical font metrics used when creating fonts. It is defined in the file \INCLUDE\PMFONT.H.

A programmer using a font in an application should use a structure that reflects the logical view of the font. An example of such a structure is FONTMETRICS, and is described in the OS/2 Version 1.2 Bindings Reference books.

Note: In this section, bits are ordered from left to right, starting from 0.

```
typedef struct _FOCAMETRICS {
    ULONG   ulIdentity;
    ULONG   ulSize;
    CHAR    szFamilyname[32];
    CHAR    szFacename[32];
    SHORT   usRegistryId;
    SHORT   usCodePage;
    SHORT   yEmHeight;
    SHORT   yXHeight;
    SHORT   yMaxAscender;
    SHORT   yMaxDescender;
    SHORT   yLowerCaseAscent;
    SHORT   yLowerCaseDescent;
    SHORT   yInternalLeading;
    SHORT   yExternalLeading;
    SHORT   xAveCharWidth;
    SHORT   xMaxCharInc;
    SHORT   xEmInc;
    SHORT   yMaxBaselineExt;
    SHORT   sCharSlope;
    SHORT   sInlineDir;
    SHORT   sCharRot;
    USHORT  usWeightClass;
    USHORT  usWidthClass;
    SHORT   xDeviceRes;
    SHORT   yDeviceRes;
    SHORT   usFirstChar;
    SHORT   usLastChar;
    SHORT   usDefaultChar;
    SHORT   usBreakChar;
    SHORT   usNominalPointSize;
    SHORT   usMinimumPointSize;
    SHORT   usMaximumPointSize;
    SHORT   fsTypeFlags;
    SHORT   fsDefn;
    SHORT   fsSelectionFlags;
    SHORT   fsCapabilities;
    SHORT   ySubscriptXSize;
    SHORT   ySubscriptYSize;
    SHORT   ySubscriptXOffset;
    SHORT   ySubscriptYOffset;
    SHORT   ySuperscriptXSize;
    SHORT   ySuperscriptYSize;
    SHORT   ySuperscriptXOffset;
    SHORT   ySuperscriptYOffset;
    SHORT   yUnderscoreSize;
    SHORT   yUnderscorePosition;
    SHORT   yStrikeoutSize;
    SHORT   yStrikeoutPosition;
    SHORT   usKerningPairs;
    SHORT   usKerningTracks;
    PSZ     pszDeviceNameOffset;
  } FOCAMETRICS;
  typedef FOCAMETRICS FAR *PFOCAMETRICS;
```

In the definitions that follow, lengths are defined in pels for bit-mapped fonts and in logical units (normalized to the xDeviceRes and yDeviceRes) for outline fonts.

The metrics data has the following meanings:

lIdentity 4-byte integer.

Must be equal to 1.

lSize 4-byte integer. Size of structure.

The size of the structure in bytes.

szFamilyname 32-character string. Family name.

The family name of the font, the basic appearance of the font. For example, Swiss.

szFacename 32-character string. Typeface name.

The full typeface name by which the font is known, for example, Helv Bold Italic. The field must contain a valid (non-NULL) facename.

idRegistry 2-byte integer. Registry identifier.

The Registry number for the font. This field is set to 0 for OS/2 Version 1.2 fonts.

usCodePage 2-byte integer. Code-page global identifier.

Defines the code page supported by the font. Code pages are described in Chapter 28, "Code Pages." This field is initially set to 850 for OS/2 Version 1.2 fonts.

Where a font contains special symbols for which there is no registered code page, then code page 65400 should be used.

yEmHeight 2-byte integer. Cap-M height.

The (nominal) height above the baseline for uppercase characters.

yXHeight 2-byte integer. X-height.

The (nominal) height above the baseline for lowercase characters (ignoring ascenders).

yMaxAscender 2-byte integer. Maximum ascender height.

The maximum height above the baseline reached by any part of any symbol in the font.

yMaxDescender 2-byte integer. Maximum descender depth.

The maximum depth below the baseline reached by any part of any symbol in the font.

yLowerCaseAscent 2-byte integer. Maximum lowercase ascender height.

The maximum height above the baseline reached by any part of any lowercase (Latin a – z) symbol in the font.

yLowerCaseDescent 2-byte integer. Maximum lowercase descender depth.

The maximum depth below the baseline reached by any part of any lowercase (Latin a – z) symbol in the font.

yInternalLeading 2-byte integer. Internal leading.

Specifies the amount of white space normally separating lines of text where lines are spaced by the maximum baseline extent. This space may contain accents.

yExternalLeading 2-byte integer. External leading.

Specifies the amount of white space in addition to external leading to be left between lines. This value may be zero.

xAveCharWidth 2-byte integer. Average weighted escapement.

The average character width is calculated according to this formula:

For the lowercase letters only, sum the individual character widths multiplied by the following weighting factors and then divide by 1000. For example:

Letter	Weight Factor
a	64
b	14
c	27
d	35
e	100
f	20
g	14
h	42
i	63
j	3
k	6
l	35
m	20
n	56
o	56
p	17
q	4
r	49
s	56
t	71
u	31
v	10
w	18
x	3
y	18
z	2
space	166

xMaxCharInc 2-byte integer. Maximum character increment.

The maximum character increment for the font.

xEmInc 2-byte integer. Em-increment.

Typically, the increment of the graphic character "M."

yMaxBaselineExt 2-byte integer. Maximum baseline extent.

The maximum space occupied by the font (typically, the sum of the maximum ascender and maximum descender).

usCharSlope 2-byte integer. Nominal character slope.

Defines the nominal slope for the characters of a font. The slope is defined in degrees and minutes increasing clockwise from the vertical. An *Italic* font is an example of a font with a nonzero slope.

Programming Note: This angle is contained in a two-part unsigned number. The first 9 bits contain the number of degrees by which the baseline is rotated (a number in the range 0 through 359), and the next 6 bits contain the number of minutes (a number in the range 0 through 59). The final bit is reserved 0. Values outside the specified ranges are invalid.

usInlineDir 2-byte integer. Inline (escapement) direction.

The direction in which the characters in the font are designed for viewing, in degrees increasing clockwise from the horizontal (left-to-right). Characters are added to a line of text along the character baseline in the inline direction. For example, a Hebrew font is written from right to left and has an inline direction of 180 degrees.

Programming Note: This angle is contained in a two-part unsigned number. The first 9 bits contain the number of degrees by which the baseline is rotated (a number in the range 0 through 359), and the next 6 bits contain the number of minutes (a number in the range 0 through 59). The final bit is reserved 0. Values outside the specified ranges are invalid.

usCharRot 2-byte integer. Character rotation.

The rotation of the character glyph with respect to the baseline.

Programming Note: This angle is contained in a two-part unsigned number. The first 9 bits contain the number of degrees by which the baseline is rotated (a number in the range 0 through 359), and the next 6 bits contain the number of minutes (a number in the range 0 through 59). The final bit is reserved 0. Values outside the specified ranges are invalid.

usWeightClass 2-byte integer. Weight class.

Indicates the visual weight (thickness of strokes) of the characters in the font:

Value	Description
1	Ultra-light
2	Extra-light
3	Light
4	Semi-light
5	Medium (normal)
6	Semi-bold
7	Bold
8	Extra-bold
9	Ultra-bold.

usWidthClass 2-byte integer. Width class.

Indicates the relative aspect ratio of the characters of the font in relation to the "normal" aspect ratio for this type of font:

Value	Description	% of normal
1	Ultra-condensed	50
2	Extra-condensed	62.5
3	Condensed	75
4	Semi-condensed	87.5
5	Medium (normal)	100
6	Semi-expanded	112.5
7	Expanded	125
8	Extra-expanded	150
9	Ultra-expanded	200

xDeviceRes 2-byte integer. Device (design) resolution x.

The x resolution of the intended target device. For raster fonts the units are pels per inch and for outline fonts the units are the logical units defining the drawing grid.

yDeviceRes 2-byte integer. Device (design) resolution y.

The y resolution of the intended target device. For raster fonts the units are pels per inch and for outline fonts the units are the logical units defining the drawing grid.

usFirstChar 2-byte integer. First character.

The code point of the first character in the font.

usLastChar 2-byte integer. Last character.

The code point of the last character in the font, expressed as an offset from **usFirstChar**.

All code points between the first and last character specified must be supported by the font.

usDefaultChar 2-byte integer. Default character.

The code point that is used if a code point outside the range supported by the font is used, expressed as an offset from **usFirstChar**.

It is recommended that the default character should be the block character, glyph 219.

usBreakChar 2-byte integer. Break character.

The code point that represents the "space" or "break" character for this font, expressed as an offset from **usFirstChar**. For example, if the first character is the space in code page 850, usFirstChar=32 and usBreakChar=0.

usNominalPointSize 2-byte integer. Nominal point size.

The height of the font specified in decipoints (one 720th of an inch). The nominal size is the size for which the font is designed.

usMinimumPointSize 2-byte integer. Minimum point size.

The minimum height, in decipoints, to which the font may be scaled down for display to retain font fidelity.

usMaximumPointSize 2-byte integer. Maximum point size.

The maximum height, in decipoints, to which the font may be scaled up for display to retain font fidelity.

fsType 2 bytes of bits. Type flags.

Contains the following information:

FMETRICS_FIXED
Characters in the font have the same fixed width.

FMETRICS_LICENSED
Licensed (protected) font.

FMETRICS_KERNED
Font contain kerning information.

FMETRICS_ANTI_ALIASED
Font contains more than one plane.

FMETRICS_64K
Font is larger than 64K bytes in size.

If the following two bits are false than the font is for single-byte code pages. One of the bits may be set.

FMETRICS_DOUBLE_BYTE
Font for double-byte code pages.

FMETRICS_MIXED_BYTE
Font for mixed single/double-byte code pages.

fsDefn 2 bytes of flags. Font definition flags.

Contains the following font-definition data:

FMETRICS_OUTLINE
Font is a vector (outline) font, otherwise it is a bit-map font.

FMETRICS_GENERIC
Font is in a format that can be used by the GPI, otherwise it is a device font.

fsSelection 2 bytes of bits. Font selection flags.

Contains information concerning the nature of the font patterns, as follows:

FMETRICS_ITALIC
Font contains Italic characters, otherwise they are upright.

FMETRICS_UNDERSCORE
Characters are underscored.

FMETRICS_NEGATIVE
Characters have their foreground and background reversed.

FMETRICS_OUTLINED
Outline (hollow) characters, otherwise they are solid.

FMETRICS_OVERSTRUCK
Characters are overstruck.

fsCapabilities 2-byte integer. Capabilities flags.

This set of flags tells the engine which simulations can be performed on the font:

FMETRICS_NOMIX
Characters may be mixed with graphics.

ySubscriptXSize 2-byte integer. Subscript horizontal font size.

The recommended horizontal point size for subscripts for this font.

ySubscriptYSize 2-byte integer. Subscript vertical font size.

The recommended vertical point size for subscripts for this font.

ySubscriptXOffset 2-byte integer. Subscript x offset.

The recommended vertical offset for subscripts for this font.

ySubscriptYOffset 2-byte integer. Subscript y offset.

The recommended horizontal offset for subscripts for this font.

ySuperscriptXSize 2-byte integer. Superscript horizontal font size.

The recommended horizontal point size for superscripts for this font.

ySuperscriptYSize 2-byte integer. Superscript vertical font size.

The recommended vertical point size for superscripts for this font.

ySuperscriptXOffset 2-byte integer. Superscript x offset.

The recommended vertical offset for superscripts for this font.

ySuperscriptYOffset 2-byte integer. Superscript y offset.

The recommended horizontal offset for superscripts for this font.

yUnderscoreSize 2-byte integer. Underscore width.

Width of the underscore stroke.

yUnderscorePosition 2-byte integer. Underscore position.

The position of the underscore stroke from the baseline.

yStrikeoutSize 2-byte integer. Throughscore size.

Width of the strikeout stroke.

yStrikeoutPosition 2-byte integer. Throughscore position.

The position of the strikeout stroke relative to the baseline.

usKerningPairs 2-byte integer. Number of kerning pairs.

The number of kerning pairs in the kerning pair table. If the **usKerningPairs** field is zero, there is no kerning pair table.

usReserved 2-byte integer.

This is a reserved field.

pszDeviceNameOffset 4-byte integer. Device name offset.

This is an offset from the beginning of the resource to a null-terminated string with the name of the device. The mapper can tell if this font belongs to a specific device. If a font is generic, this field should be zero.

Font Character Definitions

Two formats of font character definition are supported. These are:

- Image format.

 The character glyphs are represented as pel images.

- Outline format.

 The character glyphs are represented by vector data that traces the outline of the character.

The definition consists of a header portion and a portion carrying the characters themselves.

The header portion contains information about the format of the character definitions and data about each character including width data and the offset into the definition section at which the character definition begins. (See "a-space, b-space, c-space" on page C-14.)

1. Proportional characters (a+b+c = character increment) for each character:

 $a,b,c \geq 0$

2. Characters where a, b, and c are definitions for all characters:

 $b \geq 0$
 a, c any integer

Raster fonts contain a "null character." The character definition record for this occurs after the one for the last character. Thus the format has usLastChar+2 characters, although the null character is not counted in the range returned. The null character is composed of zeros and is always eight pels wide.

Font Definition Header

This structure defines the format or the character definition records that follow it:

```
typedef struct_FONTDEFINITIONHEADER {
 ULONG        ulIdentity;
 ULONG        ulSize;
 SHORT        fsFontdef;
 SHORT        fsChardef;
 SHORT        usCellSize;
 SHORT        xCellWidth;
 SHORT        yCellHeight;
 SHORT        xCellIncrement;
 SHORT        xCellA;
 SHORT        xCellB;
 SHORT        xCellC;
 SHORT        pCellBaseOffset;
 } FONTDEFINITIONHEADER;
typedef FONTDEFINITIONHEADER FAR *PFONTDEFINITIONHEADER;
```

ulIdentity 4 bytes.

Must be equal to 2.

ulSize 4 bytes.

Size of this structure in bytes.

fsFontdef

2 bytes of flags.

Indicates which fields are present in the font definition data in the header.

Type 1

Bit 0 1 = width defined in header
Bit 1 1 = height defined in header
Bit 2 1 = char increment same as width, so that it is defined for the whole font
Bit 3 0 = a-space not defined
Bit 4 0 = b-space not defined
Bit 5 0 = c-space not defined
Bit 6 1 = base offset same for all characters.

Type 2

Bit 0 0 = width for each character unique
Bit 1 1 = height defined in header
Bit 2 0 = char increment same as width, so that it is unique for each character
Bit 3 0 = a-space not defined
Bit 4 0 = b-space not defined
Bit 5 0 = c-space not defined
Bit 6 1 = base offset same for all characters.

Type 3

Bit 0 0 = width for each character unique
Bit 1 1 = height defined in header
Bit 2 0 = char increment same as width, so that it is unique
Bit 3 0 = a-space not defined
Bit 4 0 = b-space not defined
Bit 5 0 = c-space not defined
Bit 6 1 = base offset same for all characters.

FsChardef

2 bytes of flags.

Indicates which fields are present on a per character basis.

Type 1

Bit 0 1 = width defined for each character (performance op)
Bit 1 0 = height is in header
Bit 2 0 = char increment is in header
Bit 3 0 = a-space not defined
Bit 4 0 = b-space not defined
Bit 5 0 = c-space not defined
Bit 6 0 = base offset defined in header
Bit 7 1 = offset to glyph defined.

Type 2

Bit 0 1 = width defined for each character
Bit 1 0 = height is in header
Bit 2 0 = char increment same as width
Bit 3 0 = a-space not defined
Bit 4 0 = b-space not defined
Bit 5 0 = c-space not defined
Bit 6 0 = base offset defined in header
Bit 7 1 = offset to glyph defined.

Type 3

Bit 0 1 = width not defined, use a, b, c
Bit 1 0 = height is in header
Bit 2 0 = char increment same as width
Bit 3 1 = a-space defined
Bit 4 1 = b-space defined
Bit 5 1 = c-space defined
Bit 6 0 = base offset defined in header
Bit 7 1 = offset to glyph defined.

usCellSize

2-byte integer.

Indicates the length in bytes of each character definition record (the per character data).

Type 1 6 bytes
Type 2 6 bytes
Type 3 10 bytes.

xCellWidth

2-byte integer

The width of the characters, in pels for image fonts, and relative units for outline fonts.

Type 1 Width of the characters
Type 2 Zero
Type 3 Zero.

yCellHeight

2-byte integer.

The height of the characters, in pels for image fonts, and relative units for outline fonts.

Type 1 Height of the characters
Type 2 Height of the characters
Type 3 Height of the characters.

xCellIncrement

2-byte integer.

The distance along the character baseline required to step from one character to the next (when forming a character string).

Type 1 Width of the characters
Type 2 Zero
Type 3 Zero.

xCellA

2-byte signed integer.

The width of the space before a character in the inline direction (the a-space).

Type 1 Zero
Type 2 Zero
Type 3 a-space for all characters.

xCellB

2-byte integer.

The width of a character (inline direction). The b-space.

Type 1 Zero
Type 2 Zero
Type 3 b-space for all characters.

xCellC

2-byte signed integer.

The width of the space after a character in the inline direction (the c-space).

Type 1 Zero
Type 2 Zero
Type 3 c-space for all characters.

pCellBaseOffset

2-byte signed integer.

The position of the top of a character definition relative to the baseline in the direction perpendicular to the baseline.

Type 1 Baseline offset for all characters
Type 2 Baseline offset for all characters
Type 3 Baseline offset for all characters.

Character Definition Record

xCellSize bytes per record.

The following fields may or may not be present, according to the font character definition fields flags. If a field is present, it is present for *each* character and the value applies to that character only.

There are usLastChar+2 such records for raster fonts. The final one is for the null character.

- Character Definition Offset: 4-byte integer.

 The offset into the Font Definition data at which the character definition begins.

 Data for a single character raster or vector should not span two segments; that is, if a character is too big to fit into a segment it should be put in the next segment.

 This field should be set to zero if the character being defined is a blank character.

- Character Cell Width: 2-byte integer.

 The width of the character definition in pels.

- Character Cell Height: 2-byte integer.

 The height of the character definition in pels.

- Character Increment: 2-byte integer.

 The length along the character baseline required to step from this character to the next (when forming a character string).

- Character a-space: 2-byte signed integer.

 The width of the space before the character in the inline direction.

- Character b-space: 2-byte integer.

 The width of the character shape (inline direction).

- Character c-space: 2-byte signed integer.

 The width of the space after the character in the inline direction.

- Character Baseline Offset: 2-byte signed integer.

 The position of the top of a character definition relative to the baseline in the direction perpendicular to the baseline.

Programming Note: Type 1 fonts have offset/width pairs (like type 2); however, the usCellSize and xCellIncrement are nonzero. In the fsType field of the font metrics, the proportional-space flag, bit 0, is set.

Image Data Format

The bits for each character are stored separately, and start on a byte boundary. Sequential bytes represent vertical pieces of the character image. For example, a 15-bit-wide H is stored as follows:

```
byte                    byte

  1  00000000 0000000-   13      Bytes 1 through 12 are composed of
  2  01100000 0000110-   14      whole bytes of data stored row by row.
  3  01100000 0000110-   15
  4  01100000 0000110-   16      Bytes 13 through 24 are composed of
  5  01100000 0000110-   17      bytes stored row by row, where each byte
  6  01111111 1111110-   18      contains 7 bits of information and the
  7  01111111 1111110-   19      last bit is unused.
  8  01100000 0000110-   20
  9  01100000 0000110-   21      Thus the character is laid down in
 10  01100000 0000110-   22      byte-wide columns.
 11  01100000 0000110-   23
 12  00000000 0000000-   24
```

Notes:

1. There is always an additional (null) character defined in an Image Font (defined at character position LastChar+2) which is an 8-bit wide height of the font character, set to all zeros.
2. The maximum size of each individual Image Font must not exceed 64 KB.

Fonts supplied with OS/2 Version 1.2

The following fonts are provided with OS/2 Version 1.2:

Family	Point sizes
Courier	8, 10 and 12
Roman	8, 10, 12, 14, 18 and 24
Swiss	8, 10, 12, 14, 18 and 24

System fonts are supplied in the following cell sizes:

Cell	Screen	Ratio
8x 8	CGA	2:1
8x10	EGA	1.33:1
8x12	EGA	1.33:1
8x14	VGA	1:1
8x16	VGA	1:1
12x20	8514	1:1

Outline Data Format

Fonts defined by outlines (vectors) may contain any of these graphics orders:

- Line at given position (GLINE)
- Line at current position (GCLINE)
- Relative line at given position (GRLINE)
- Relative line at current position (GCRLINE)
- Fillet at given position (GFLT)
- Fillet at current position (GCFLT)
- Sharp fillet at given position (GSFLT)
- Sharp fillet at current position (GCSFLT)
- Bézier curve at given position (GBEZ)
- Bézier curve at current position (GCBEZ)
- No operation (GNOP1)

- Comment (GCOMT)
- End of symbol definition (GESD).

The maximum length of the data in these orders is 255 bytes. The drawing order code and the length fields are not included in the length count.

The size of each outline font definition must not be longer than 64 KB.

The Kerning Pair Table

The kerning pair table record is not present if the _KerningPairs record in the metrics is zero. If it is present, the code points are words, not bytes. This table should be sorted by kpChar1 and kpChar2 order to allow binary searches.

```
typedef struct _KERNPAIRTABLE {
        ULONG   ulIdentity;
        ULONG   ulSize;
        CHAR    cFirstpair;
}KERNPAIRTABLE;

typedef struct _KERNPAIRS {
        SHORT   sFirstChar;
        SHORT   sSecondChar;
        SHORT   sKerningAmount;
}KERNINGPAIRS;
```

where:

```
ulIdentity      = 3
ulSize          = 10
sFirstChar      = First character of the kerning pair
sSecondChar     = Second character of the kerning pair
sKerningAmount  = Kerning value.  Positive values increase the
                  inter-character spacing while negative values
                  bring the characters closer together.
```

Font Directory

This section describes the directory section of a font resource. A font resource contains a directory consisting of a set of structures each containing the metrics of a font and a pointer to the font itself. This font directory is generated by the resource compiler.

The format of the font directory is:

```
typedef struct {
   USHORT      usHeaderSize;
   USHORT      usnFonts;
   USHORT      usiMETRICS;
   FONTENTRY   fntEntry[1];
} FONTDIRECTORY;

typedef struct {
   USHORT      usIndex;
   FOCAMETRICS foca;
} FONTENTRY;
```

The parameters have the following meaning:

usHeaderSize The size of the header, in bytes.

usnFonts The number of fonts in the resource.

usiMetrics	The size of the FOCAMETRICS structures that follow the header. Note that the set of metrics for all the fonts in the resource follow the header.
usIndex	The index of a particular font; an identifier assigned to the font when the resource was created (defined in the .RC file).
foca	The font metrics structure for the font.

Definitions of Terms Used When Describing Fonts

a-space, b-space, c-space
The a-space is the distance from the left of the character frame to the left edge of the character. The b-space is the width of the character. The c-space is the distance from the right edge of the character to the right of the character frame. Negative values of a and c allow adjacent character frames to overlap. See also *character increment*, and *space default values*.

average char width
The average horizontal distance from the left edge of one character to the left edge of the next. Contrast with *max char increment*.

baseline
The line on which the bottom of a character rests, and below which a descender extends.

break char code point
The *code point* of the space or break character. Contrast with *default char code point*, *first char code point*, and *last char code point*.

character increment
A set of three values (*a-space*, *b-space*, and *c-space*) that define the proportions of a character. The sum of the three values (a+b+c) specifies only one value for the entire character increment. See also *font width* and *space default values*.

character rotation
The angle by which each character is rotated around its own center, increasing clockwise from vertical. Contrast with *character slope* and *inline direction*.

character slope
The angle by which a character is slanted, increasing clockwise from vertical. Contrast with *character rotation* and *inline direction*.

default char code point
The *code point* of the character to be used if a *code point* outside the range of a font is passed to an application using that font. Contrast with *break char code point*, *first char code point*, and *last char code point*.

em height
The maximum distance above the *baseline* reached by an uppercase symbol. Contrast with *x height*.

external leading
The vertical distance from the bottom of one character to the top of the character below it. Contrast with *internal leading* and *max baseline extent*.

first char code point
The *code point* of the first character. All numbers between the *first char code point* and the *last char code point* must represent a character in the font. Contrast with *break char code point*, *default char code point*, and *last char code point*.

fixed spacing
The same amount of space separates each character. Contrast with *proportional spacing*.

font weight
The line-thickness of a character relative to its size. Contrast with *font width*.

font width
The relative width of a character to its height; condensed fonts are very narrow while expanded fonts are very wide. See also *character increment*. Contrast with *font weight*.

inline direction
The angle of a line of type, increasing clockwise from horizontal. Contrast with *character rotation* and *character slope*.

internal leading
The vertical distance from the top or bottom of a character to any accent marks that may appear with it. Contrast with *external leading*.

last char code point
The *code point* of the last character. All numbers between the *first char code point* and the *last char code point* must represent a character in the font. Contrast with *break char code point*, *default char code point*, and *first char code point*.

lowercase ascent
The maximum distance above the *baseline* reached by any part of any lowercase character. Contrast with *maximum ascender* and *x height*.

lowercase descent
The maximum distance below the *baseline* reached by any part of any lowercase character. Contrast with *maximum descender*.

max baseline extent
The maximum space occupied by the font (typically, the sum of the *maximum ascender* and *maximum descender*). Contrast with *external leading* and *max char increment*.

max char increment
The maximum horizontal distance from the left edge of one character to the left edge of the next character to the right. Contrast with *average char width* and *max baseline extent*.

maximum ascender
The maximum distance that any part of any character may extend above the *x height* of a font. Contrast with *lowercase ascent* and *maximum descender*.

maximum descender
The maximum distance that any part of any character may extend below the *x height* of a font. Contrast with *lowercase descent* and *maximum ascender*.

maximum vert point size
The maximum vertical dimensions to which a font can be resized. Contrast with *minimum vert point size* and *nominal vert point size*.

minimum vert point size
The minimum vertical dimensions to which a font can be resized. Contrast with *maximum vert point size* and *nominal vert point size*.

nominal vert point size
The normal display size of a font. Contrast with *maximum vert point size* and *minimum vert point size*.

pel
The smallest element of a display surface that can be independently assigned color and density.

point
Printer's unit of measurement. There are 72 points to an inch (approximately 3.5 points to a millimeter).

proportional spacing
The space that each character occupies is in proportion to its width. See also *font width*. Contrast with *fixed spacing*.

Registry ID
A code number that Presentation Manager uses to register a font file as a resource.

space default values
Values that specify the space to be left between characters. Once defined, they are used for the entire font, and do not have to be specified for each character. However, they can be changed for characters that require more or less spacing than the defaults provide, by giving values for the *a Space* and the *c Space*. See also *character increment*.

strikeout position
The distance of the strikeout character above the *baseline* (in *pels*). See also *strikeout size* and *underscore position*.

strikeout size
The size of the strikeout character (in *points*). See also *strikeout position* and *underscore size*.

subscript position
The distance of a subscript character of a font below the *baseline* (in *pels*). See also *subscript size* and *superscript position*.

subscript size
The size of a subscript character (in *points*). See also *subscript position* and *superscript size*.

superscript position
The distance of a superscript character above the *baseline* (in *pels*). See also *subscript position* and *superscript size*.

superscript size
The size of a superscript character (in *points*). See also *subscript size* and *superscript position*.

target dev resolution X
The number of *pels* per inch in the horizontal axis of a display device on which a font is to be displayed. Contrast with *target dev resolution Y*.

target dev resolution Y
The number of *pels* per inch in the vertical axis of a display device on which a font is to be displayed. Contrast with *target dev resolution X*.

underscore position
The distance in *pels* of the first underscore stroke from the *baseline* of a font. Successive strokes below this create a heavier underscore. See also *strikeout position* and *underscore size*.

underscore size
The size of the underscore character measured in single strikeout strokes. See also *strikeout size* and *underscore position*.

x height
The maximum distance above the *baseline* reached by a lowercase character. Contrast with *em height* and *lowercase ascent*.

Appendix D. Format of Interchange Files

Metafile Restrictions

This section lists some general rules that must be followed when creating a metafile that is to be acceptable to SAA-conforming implementations, or replayed into a presentation space that is in **draw-and-retain** or **retain** mode (see GpiSetDrawingMode).

- These items must be established or defaulted before any drawing occurs to the graphics presentation space:
 - The graphics field (GpiSetGraphicsField). For an SAA-conforming metafile, the graphics field must be defaulted or set to no clipping.
 - The code page for the default character set (GpiSetCp).
 - The color table (GpiCreateLogColorTable). The size of the color table must not exceed 31KB.
 - the default viewing transform (GpiSetDefaultViewMatrix).
 - The setting of the draw controls (GpiSetDrawControl). DCTL_DISPLAY must be defaulted or set on.
 - The default values of attributes (see GpiSetDefAttrs), viewing limits (see GpiSetDefViewingLimits), primitive tag (see GpiSetDefTag) and arc parameters (see GpiSetDefArcParams).

 These items must not be changed while the metafile context remains open.

- These calls must not be used:
 - GpiBitBlt
 - GpiDeleteSetId (note that this means that local identifiers cannot be used again within the picture)
 - GpiErase
 - GpiExcludeClipRectangle
 - GpiIntersectClipRectangle
 - GpiOffsetClipRegion
 - GpiPaintRegion
 - GpiResetPS
 - GpiSetClipRegion
 - GpiSetPel
 - GpiSetPS
 - DevEscape (for an escape which is metafiled).

- The metafile context must not be reassociated.

- If a bit map is used as the source of a GpiWCBitBlt operation, or as an area-fill pattern, it must not be modified.

- Only these foreground mixes must be used (see GpiSetMix):
 - FM_DEFAULT
 - FM_OR
 - FM_OVERPAINT
 - FM_LEAVEALONE.

- Only these background mixes must be used (see GpiSetBackMix):
 - BM_DEFAULT
 - BM_OVERPAINT
 - BM_LEAVEALONE.

Programming Note: There is no restriction concerning the use of primitives outside segments. These are metafiled in segment(s) with zero identifier.

Metafile Data Format

This section describes the format of the data in a metafile, as it would be stored in an OS/2 Version 1.2 disk file.

Metafile data is stored as a sequence of **structured fields**. Each structured field starts with an eight-byte header consisting of a two-byte **length** field and a three-byte **identifier** field. These are followed by a one-byte **flags** field and a two-byte **segment sequence number** field.

The length field contains a count of the total number of bytes in the structured field, including the length field. The identifier field uniquely identifies the type of the structured field.

The flags and segment sequence number fields are always zero.

Following the header are positional parameters that are optional and dependent on the particular structured field.

Following the positional parameters are non-positional parameters called *triplets*. These are self-defining parameters and consist of a one-byte **length** field, followed by a one-byte **identifier** field, followed by the data of the parameter.

The length field contains a count of the total number of bytes in the triplet, including the length and identifier fields. The identifier field identifies uniquely the type of the triplet.

A metafile is structured into a number of different functional components; for example, document and graphics object. Each component comprises a number of structured fields, and is delimited by "begin-component" and "end-component" structured fields. Structured fields marked as **required**, inside an **optional** structured field bracket, are required if the containing bracket is present.

The graphics orders that describe a picture occur in the **graphics data** structured field; see page D-15.

Structured Field Formats

The format of the various structured fields is given below:

Begin Document

Structured Field Introducer (BDT): required

```
0-1   Length X'n+1E'
2-4   BDT X'D3A8A8'
5     Flags X'00'
6-7   Segment sequence number X'0000'
```

Parameters

```
0-7   Document name   C'0000 0001'
8     Architecture version X'00'
9     Document security X'00'
```

Triplets (all required)

```
0     Length X'05'
1     Triplet Id X'18'
2     Interchange set type X'03' (resource document)
3-4   Base set definition X'0C00' (level 12, version 0)

0     Length X'06'
1     Triplet Id X'01'
2-5   GCID

0     Length X'n+1'
1     Triplet Id X'65'
2-n   Comment, used for metafile description of
      up to 252 bytes.
```

Begin Resource Group (BRG): required

Structured Field Introducer

```
0-1   Length X'0010'
2-4   BRG X'D3A8C6'
5     Flags X'00'
6-7   Segment sequence number X'0000'
```

Parameters

```
0-7   Resource group name   C'0000 0002'
```

Begin Color Attribute (BCA) Table: required

Structured Field Introducer

```
0-1   Length X'0010'
2-4   BCA X'D3A877'
5     Flags X'00'
6-7   Segment sequence number X'0000'
```

Parameters

```
0-7   Color table name C'0000 0004'
```

Color Attribute Table (CAT): required

Structured Field Introducer

```
0-1   Length X'n+8'
2-4   CAT X'D3B077'
5     Flags X'00'
6-7   Segment sequence number X'0000'
```

Parameters

Base Part (required)
```
0  Flags
   0   Reserved B'0'
   1   Reset
       B'0' Do not reset to default
       B'1' Do reset to default
   2-7 Reserved B'000000'
1  Reserved X'00'
2  LCTID X'00'
```

Element list(s) (triple generating) are mutually-exclusive. One or other is required.

Element List (repeating)
```
0     Length of this parameter
1     Type X'01': element list
2     Flags X'00': reserved
3     Format
      X'01' RGB
4-6   Starting Index (Top Byte Truncated)
7     Size of RGB component1 X'08'
8     Size of RGB component2 X'08'
9     Size of RGB component3 X'08'
10    Number of bytes in each following color
      triple X'04'
11-m  Color triples
```

Triple Generating
```
0     Length of this parameter X'0A'
1     Type X'02': bit generator
2     Flags
      0 ABFlag
        B'0' Normal
      1-7 Reserved B'0000000'
3     Format
      X'01' RGB
4-6   Starting index (top byte truncated)
7     Size of RGB component1 X'08'
8     Size of RGB component2 X'08'
9     Size of RGB component3 X'08'
```

End Color Attribute (ECA) Table: required

Structured Field Introducer
```
0-1   Length X'0010'
2-4   ECA X'D3A977'
5     Flags X'00'
6-7   Segment sequence number X'0000'
```

Parameters
```
0-7   Color table name C'0000 0004'
```

Begin Image Object (BIM): optional, repeating

Structured Field Introducer
```
0-1   Length X'0010'
2-4   BIM X'D3A8FB'
5     Flags X'00'
6-7   Segment sequence number X'0000'
```

Parameters
```
0-7   Image name C'xxxx xxxx'
```

Begin Resource Group (BRG): optional

Structured Field Introducer
0-1 Length X'0010'
2-4 BRG X'D3A8C6'
5 Flags X'00'
6-7 Segment sequence number X'0000'

Parameters
0-7 Resource group name C'xxxx xxxx'

Begin Color Attribute Table (BCA): optional

Structured Field Introducer
0-1 Length X'0010'
2-4 BCA X'D3A877'
5 Flags X'00'
6-7 Segment sequence number X'0000'

Parameters
0-7 Color table name C'xxxx xxxx'

Color Attribute Table (CAT): required

Structured Field Introducer
0-1 Length
2-4 CAT X'D3B077'
5 Flags X'00'
6-7 Segment sequence number X'0000'

Parameters

Base Part
0 Flags X'00'
1 Reserved X'00'
2 LUTID

Element List (repeating)
0 Length of this parameter
1 Type X'01': element list
2 Flags X'00': reserved
3 Format X'01': RGB
4-6 Starting index (top byte truncated)
7 Size of RGB component1 X'08'
8 Size of RGB component2 X'08'
9 Size of RGB component3 X'08'
10 Number of bytes in each following color triple X'03'
11-n Color triples

End Color Attribute Table (ECA): required if BCA present

Structured Field Introducer
0-1 Length X'0010'
2-4 ECA X'D3A977'
5 Flags X'00'
6-7 Segment sequence number X'0000'

Parameters
0-7 Color Table name C'xxxx xxxx'

End Resource Group (ERG): required if BRG present

Structured Field Introducer
0-1 Length X'0010'
2-4 ERG X'D3A9C6'
5 Flags X'00'
6-7 Segment sequence number X'0000'

Parameters
0-7 Resource Group name C'xxxx xxxx'

Begin Object Environment Group (BOG): optional

Structured Field Introducer
0-1 Length X'0010'
2-4 BOG X'D3A8C7'
5 Flags X'00'
6-7 Segment sequence number X'0000'

Parameters
0-7 Object environment group name C'xxxx xxxx'

Map Color Attribute (MCA) Table: required

Structured Field Introducer
0-1 Length X'001A'
2-4 MCA X'D3AB77'
5 Flags X'00'
6-7 Segment sequence number X'0000'

Parameters
0-1 Length

Triplet (required)
0 Length X'0C'
1 Triplet type: fully qualified name X'02'
2 Type: ref to Begin Resource Object X'84'
3 ID X'00'
4-11 Color table name C'xxxx xxxx'

lcid (required)
0 Length X'04'
1 Triplet type: resource local ID X'24'
2 Type color table resource X'07'
3 Local identifier (LUT-ID) X'01'

End Object Environment Group (EOG): required if BOG present

Structured Field Introducer
0-1 Length X'0010'
2-4 EOG X'D3A9C7'
5 Flags X'00'
6-7 Segment sequence number X'0000'

Parameters
0-7 Object Environment Group name C'xxxx xxxx'

Image Data Descriptor (IDD): required

Structured Field Introducer
0-1 Length X'0011'
2-4 IDD X'D3A6FB'
5 Flags X'00'
6-7 Segment sequence number X'0000'

Parameters

0 Unit of measure:
X'00' tens of inches
X'01' tens of centimeters
1-2 X resolution image points / UOM
3-4 Y resolution image points / UOM
5-6 X extent of image PS
7-8 Y extent of image PS

Image Picture Data (IPD): required

Structured Field Introducer

0-1 Length
2-4 IPD X'D3EEFB'
5 Flags X'00'
6-7 Segment sequence number X'0000'

Parameters (all required and in this order, except that only one of Image LUT-ID and IDE structure is present)

Begin Segment

0 Type X'70': begin segment
1 Length of following X'00'

Begin Image Content

0 Type X'91': Begin Image Content
1 Length of following X'01'
2 Format X'FF'

Image Size

0 Type X'94': image size
1 Length of following X'09'
2 Units of measure X'02': logical
3-4 Horizontal resolution
5-6 Vertical resolution
7-8 Height in pels
9-10 Width in pels

Image Encoding

0 Type X'95': image encoding
1 Length of following X'02'
2 Compression algorithm X'03': none
3 Recording algorithm X'03': bottom-to-top

Image IDE-Size

0 Type X'96': image IDE-Size
1 Length of following X'01'
2 Number of bits per element

Image LUT-ID

(For bit maps with other than
24 bits per pel)
0 Type X'97' Image LUT-ID
1 Length of following X'01'
2 LUT-ID

IDE Structure

(For bit maps with 24 bits per pel)
0 Type X'9B': IDE structure
1 Length of following X'08'
2 Flags:
0 ABFlag
B'0' Normal (Additive)
1-7 Reserved B'0000000'

```
3     Format
      X'01' RGB
4-6   Reserved X'000000'
7     Size of element 1
8     Size of element 2
9     Size of element 3
```

Image Picture Data (IPD): required, repeating

```
Structured Field Introducer
0-1   Length
2-4   IPD X'D3EEFB'
5     Flags X'00'
6-7   Segment sequence number X'0000'

Parameters

Image Data
0-1   Type X'FE92': image data
2-3   Length of following
4-n   Image data (scan lines of bit maps)

End Image Content
      (required, only present in last
      Image Picture Data)
0     Type X'93': End Image Content
1     Length of following X'00'

End Segment
      (required, only present in last
      Image Picture Data)
0     Type X'71': end segment
1     Length of following X'00'
```

End Image Object (EIM): required if BIM present

```
Structured Field Introducer
0-1   Length X'0010'
2-4   EIM X'D3A9FB'
5     Flags X'00'
6-7   Segment sequence number X'0000'

Parameters
0-7 Image name C'xxxx xxxx'
```

Begin Graphics Object (BGR): required

```
Structured Field Introducer
0-1   Length X'0010'
2-4   BGR X'D3A8BB'
5     Flags X'00'
6-7   Segment sequence number X'0000'

Parameters
0-7 Graphics object name C'0000 0007'
```

Begin Object Environment Group (BOG): optional

```
Structured Field Introducer
0-1   Length X'0010'
2-4   BOG X'D3A8C7'
5     Flags X'00'
6-7   Segment sequence number X'0000'

Parameters
0-7 Object Environment Group name C'0000 0007'
```

Map Color Attribute Table (MCA): required

Structured Field Introducer
0-1 Length X'0016'
2-4 MCA X'D3AB77'
5 Flags X'00'
6-7 Segment sequence number X'0000'

Parameters
0-1 Length

Triplet (required)
0 Length X'0C'
1 Triplet type: fully qualified name X'02'
2 Type: ref to Begin Resource Object X'84'
3 ID X'00'
4-11 Color table name C'0000 0004'

Map Coded Font (MCF): required, for default font

Structured Field Introducer
0-1 Length X'20'
2-4 MCF X'D3AB8A'
5 Flags X'00'
6-7 Segment sequence number X'0000'

Parameters
0-1 Length

Triplets (required)

Font name
0 Length X'0C'
1 Triplet type: fully qualified name X'02'
2 Type: ref to coded font X'84'
3 ID X'00'
4-11 Coded font name: C'nnxx xxxx'
where n is X'FF'

lcid
0 Length X'04'
1 Triplet type: Resource Local ID X'24'
2 Type: Coded Font Resource X'05'
3 Local identifier (LCID) X'00'

Font Binary GCID
0 Length X'06'
1 Triplet type: Font Binary GCID X'20'
2-5 GCID

Map Coded Font (MCF): optional, repeating, for loaded fonts

Structured Field Introducer
0-1 Length X'58'
2-4 MCF X'D3AB8A'
5 Flags X'00'
6-7 Segment sequence number X'0000'

Parameters
0-1 Length

Triplets (required)

Font name
0 Length X'0C'
1 Triplet type: fully qualified name X'02'
2 Type: ref to coded font X'84'
3 ID X'00'
4-11 Coded font name

lcid
0 Length X'04'
1 Triplet type: Resource Local ID X'24'
2 Type: coded font resource X'05'
3 Local identifier (LCID)

Font Attributes
0 Length X'14'
1 Triplet type: Font Descriptor X'1F'
2 Weight Class
3 Width Class
4-5 Font Height
6-7 Char Width
8 Descript Flags
9 Usage Codes
10 Family
11 Activity Class
12 Font Quality
13-14 CAP Height
15-16 X Height
17-18 Line Density
19 Use Flags

Font Binary GCID
0 Length X'06'
1 Triplet type: Font Binary GCID X'20'
2-5 GCID

Font Typeface
0 Length X'24'
1 Triplet type: fully qualified name X'02'
2 Type: ref to font typeface X'08'
3 ID X'00'
4-35 Font typeface C'xxx..xxx'

Map Data Resource (MDR): optional, repeating

Structured Field Introducer
0-1 Length X'1D'
2-4 MDR X'D3ABC3'
5 Flags X'00'
6-7 Segment sequence number X'0000'

Parameters
0-1 Length

Triplets (required)

Bit-map Name
0 Length X'0C'
1 Triplet type: fully qualified name X'02'
2 Type: ref to Image Object X'84'
3 ID X'00'
4-11 Image name C'xxxx xxxx'

Extended Resource lcid
0 Length X'07'
1 Triplet type: Extended Resource Local ID X'22'
2 Type: Image Resource X'10'
3-6 Bit-map handle

End Object Environment Group (EOG): required if BOG present

Structured Field Introducer
0-1 Length X'0010'
2-4 EOG X'D3A9C7'
5 Flags X'00'
6-7 Segment sequence number X'0000'

Parameters
0-7 Object Environment Group name C'0000 0007'

Graphics Data Descriptor (GDD): required

Structured Field Introducer
0-1 Length X'nnnn'
2-4 GDD X'D3A6BB'
5 Flags X'00'
6-7 Segment sequence number X'0000'

Parameters (all required and in order)

0 X'F7' Specify GVM Subset
1 Length of following data X'07'
2 X'B0' drawing order subset
3-4 X'0000'
5 X'23' Level 3.2
6 X'01' Version 1
7 Length of following field X'01'
8 Coordinate types in data
X'04' Intel16
X'05' Intel32

0 X'F6' Set Picture Descriptor
1 Length of following data
2 Flags
0 B'0' Picture in 2D
1 Picture Dimensions
B'0' Not absolute (PU_ARBITRARY PS)
B'1' Absolute (example: PU_TWIPS PS)
2 Picture Elements
B'0' Not pels
B'1' Pels (PU_PELS PS)
(Bit 1 must also be set)
3-7 B'00000'
3 X'00' Reserved
4 Picture frame size coordinate type
X'04' Intel16
X'05' Intel32
5 UnitsOfMeasure
X'00' Ten inches
X'01' Decimeter
6-11 or 6-17 (2 or 4 bytes) Resolution.
GPS Units / UOM on x axis
GPS Units / UOM on y axis
GPS Units / UOM on z axis
12-23 or 18-41 (2 or 4 bytes) Window Size.
GPS X left, X right
GPS Y bottom, Y top
GPS Z near, Z far

```
0   X'21' Set Current Defaults
1   Length of following data
2   Set Default Parameter Format X'08'
3-4 Mask X'E000'
5   Names X'8F'
6   Coordinates
    X'00' Picture in 2D
7   Transforms
    X'04' Intel16
    X'05' Intel32
8   Geometrics
    X'04' Intel16
    X'05' Intel32

0   X'21' Set Current Defaults
1   Length of following data
2   Set default viewing transform X'07'
3-4 Mask X'CC0C'
5   Names X'8F'
6-n M11, M12, M21, M22, M41, M42   Matrix
                                   elements


0   X'21' Set Current Defaults
1   Length of following data
2   Set default line attributes X'01'
3-4 Mask - OR of as many of the following
    bits as are required:
    X'8000'  Line type
    X'4000'  Line width
    X'2000'  Line end
    X'1000'  Line join
    X'0800'  Stroke width
    X'0008'  Line color
    X'0002'  Line mix
5   Flags
    X'0F' Set indicated default
                attributes to initial values.
                (Data field is not present
                in this instance).
    X'8F' Set indicated default attributes
                to specified values.
6-n Data - data values as required, in the
                following order if present.
    No space is reserved for attributes for
    which the corresponding mask flag was not set.
    (1 byte)  - Line type
    (1 byte)  - Line width
    (1 byte)  - Line end
    (1 byte)  - Line join
    (G bytes) - Stroke width
    (4 bytes) - Line color
    (1 byte)  - Line mix
              (G=2 or 4 depending on the Geometrics
               parameter of Set Default Parameter Format)

0   X'21' Set Current Defaults
1   Length of following data
2   Set Default Character Attributes X'02'
3-4 Mask - OR of as many of the following
    bits as are required:
    X'8000'  Character angle
    X'4000'  Character box
```

```
    X'2000'  Character direction
    X'1000'  Character precision
    X'0800'  Character set
    X'0400'  Character shear
    X'0008'  Character color
    X'0004'  Character background color
    X'0002'  Character mix
    X'0001'  Character background mix
5   Flags
    X'0F' Set indicated default attributes to initial values.
                  (Data field is not present in this case).
    X'8F' Set indicated default attributes to specified values.
6-n Data - data values as required, in the following order if present.
    No space is reserved for attributes for which the corresponding Mask
    flag was not set.
    (2*G bytes) - Character angle
    (2*G + 4 bytes) - Character box
    (1 byte)    - Character direction
    (1 byte)    - Character precision
    (1 byte)    - Character set
    (2*G bytes) - Character shear
    (4 bytes)   - Character color
    (4 bytes)   - Character background color
    (1 byte)    - Character mix
    (1 byte)    - Character background mix
        (G=2 or 4 depending on the Geometrics parameter of Set Default
         Parameter Format)


0   X'21' Set Current Defaults
1   Length of following data
2   Set Default Marker Attributes X'03'
3-4 Mask - OR of as many of the following bits as are required:
    X'4000'  Marker box
    X'1000'  Marker precision
    X'0800'  Marker set
    X'0100'  Marker symbol
    X'0008'  Marker color
    X'0004'  Marker background color
    X'0002'  Marker mix
    X'0001'  Marker background mix
5   Flags
    X'0F' Set indicated default attributes to initial values.
                  (Data field is not present in this instance)
    X'8F' Set indicated default attributes to specified values.
6-n Data - data values as required, in this order if present.
    No space is reserved for attributes for which the corresponding Mask
    flag was not set.
    (2*G bytes) - Marker box
    (1 byte)    - Marker precision
    (1 byte)    - Marker set
    (1 byte)    - Marker symbol
    (4 bytes)   - Marker color
    (4 bytes)   - Marker background color
    (1 byte)    - Marker mix
    (1 byte)    - Marker background mix
        (G=2 or 4 depending on the Geometrics parameter of Set Default
         Parameter Format)

0   X'21' Set Current Defaults
1   Length of following data
2   Set Default Pattern Attributes X'04'
3-4 Mask - OR of as many of the following bits as are required:
    X'0800'  Pattern set
```

```
    X'0100'  Pattern symbol
    X'0080'  Pattern reference point
    X'0008'  Pattern color
    X'0004'  Pattern background color
    X'0002'  Pattern mix
    X'0001'  Pattern background mix
5   Flags
    X'0F' Set indicated default attributes to initial values.
                 (Data field is not present in this instance)
    X'8F' Set indicated default attributes to specified values.
6-n Data - data values as required, in this order if present.
    No space is reserved for attributes for which the corresponding Mask
    flag was not set.
    (1 byte)    - Pattern set
    (1 byte)    - Pattern symbol
    (2*G bytes) - Pattern reference point
    (4 bytes)   - Pattern color
    (4 bytes)   - Pattern background color
    (1 byte)    - Pattern mix
    (1 byte)    - Pattern background mix
        (G=2 or 4 depending on the Geometrics parameter of Set Default
         Parameter Format)

0   X'21' Set Current Defaults
1   Length of following data
2   Set Default Image Attributes X'06'
3-4 Mask - OR of as many of these bits as are required:
    X'0008'  Image color
    X'0004'  Image background color
    X'0002'  Image mix
    X'0001'  Image background mix
5   Flags
    X'0F' Set indicated default attributes to initial values.
                 (Data field is not present in this instance)
    X'8F' Set indicated default attributes to specified values.
6-n Data - data values as required, in this order if present.
    No space is reserved for attributes for which the corresponding Mask
    flag was not set.
    (4 bytes)   - Image color
    (4 bytes)   - Image background color
    (1 byte)    - Image mix
    (1 byte)    - Image background mix

0   X'21' Set Current Defaults
1   Length of following data
2   Set Default Viewing Window X'05'
3-4 Mask - OR of as many of the following bits as are required:
    X'8000'  x left limit
    X'4000'  x right limit
    X'2000'  y bottom limit
    X'1000'  y top limit
5   Flags
    X'0F' Set indicated default attributes to initial values.
                 (Data field is not present in this case).
    X'8F' Set indicated default attributes to specified values.
6-n Data - data values as required, in the following order if present.
    No space is reserved for attributes for which the corresponding Mask
    flag was not set.
    (2*G bytes) - x left limit
    (2*G bytes) - x right limit
    (2*G bytes) - y bottom limit
    (2*G bytes) - y top limit
```

```
0   X'21' Set Current Defaults
1   Length of following data
2   Set Default Arc Parameters X'0B'
3-4 Mask - OR of as many of the following bits as are required:
    X'8000'  P value
    X'4000'  Q value
    X'2000'  R value
    X'1000'  S value
5   Flags
    X'0F' Set indicated default attributes to initial values.
                (Data field is not present in this case).
    X'8F' Set indicated default attributes to specified values.
6-n Data - data values as required, in the following order if present.
    No space is reserved for attributes for which the corresponding Mask
    flag was not set.
    (G bytes)   - P value
    (G bytes)   - Q value
    (G bytes)   - R value
    (G bytes)   - S value

0   X'21' Set Current Defaults
1   Length of following data
2   Set Default Pick Identifier X'0C'
3-4 Mask - OR of as many of the following bits as are required:
    X'8000'  Pick identifier
5   Flags
    X'0F' Set indicated default attributes to initial values.
                (Data field is not present in this case).
    X'8F' Set indicated default attributes to specified values.
6-n Data - data values as required, in the following order if present.
    No space is reserved for attributes for which the corresponding Mask
    flag was not set.
    (4 bytes)   - Pick identifier

0   X'E7' Set Bit-map Identifier
1   Length of following data X'07'
2-3 Usage Flags X'8000'
4-7 Bit-map handle
8   Lcid
```

Graphics Data (GAD): optional, repeating

Structured Field Introducer

```
0-1   Length X'n+9'
2-4   GAD X'D3EEBB'
5     Flags X'00'
6-7   Segment sequence number X'0000'
```

Parameters (maximum length in one structured field is 32759)

Graphics Segment (optional, repeating)

Segment data (including the Begin Segment parameter) can be split at any point between successive Graphics Data structured fields.

```
0     X'70' Begin Segment
1     Length of following data X'0E'
2-5   Segment identifier
6     Segment attributes (1)
      0 B'1' Invisible
      1 B'1' Propagate invisibility
      2 B'1' Detectable
      3 B'1' Propagate detectability
      6 B'1' Dynamic
```

```
        7 B'1' Fast chaining
   7    Segment attributes (2)
        0 B'1' Non-chained
        3 B'1' Prolog
  8-9   Segment data length (low-order 2 bytes)
 10-13  Reserved
 14-15  Segment data length (high-order 2 bytes)
 16-n   Graphics orders (see page 27-1)
```

End Graphics Object (EGR)

Structured Field Introducer

```
0-1   Length X'0010'
2-4   EGR X'D3A9BB'
5     Flags X'00'
6-7   Segment sequence number X'0000'
```

Parameters

```
0-7 Graphics object name C'0000 0007'
```

End Resource Group (ERG): required

Structured Field Introducer

```
0-1   Length X'0010'
2-4   ERG X'D3A9C6'
5     Flags X'00'
6-7   Segment sequence number X'0000'
```

Parameters

```
0-7 Resource Group name   C'0000 0002'
```

End Document (EDT): required

Structured Field Introducer

```
0-1   Length X'0010'
2-4   EDT X'D3A9A8'
5     Flags X'00'
6-7   Segment sequence number X'0000'
```

Parameters

```
0-7   Document name   C'0000 0001'
```

Appendix E. Initialization File Information

Initialization files include information about printers, queues, and system preferences set by the user from the control panel. Applications can query this information by using the WinQueryProfileData, WinQueryProfileInt, WinQueryProfileSize, and WinQueryProfileString calls.

All data in initialization files is accessed by a two-level hierarchy of application name, and key name within an application. Presentation Manager system data is keyed off "applications" that have names starting with PM_.

The application name/key name combinations that applications may need to use are listed below, together with the definition of the corresponding data.

Application name	"PM_ControlPanel"
Key name	"Beep"
Type	integer
Content/value	1 or 0.

Application name	"PM_ControlPanel"
Key name	"LogoDisplayTime"
Type	integer
Content/value	$-1 \leq$ time $\leq$ 32767 milliseconds.
	Indefinite display −1
	No display 0
	Timed display >0.

Application name	"PM_National"
Key name	"iCountry"
Type	integer
Content/value	country code:

Arabic	785
Australian	61
Belgian	32
Canadian-French	2
Danish	45
Finnish	358
French	33
German	49
Hebrew	972
Italian	39
Japanese	81
Korean	82
Latin-American	3
Netherlands	31
Norwegian	47
Portuguese	351
Simpl. Chinese	86
Spanish	34
Swedish	46
Swiss	41
Trad. Chinese	88
UK-English	44
US-English	1
Other country	0.

Application name	"PM_National"
Key name	"iDate"
Type	integer
Content/value	0 = MDY; 1 = DMY; 2 = YMD.

Application name	"PM_National"
Key name	"iCurrency"
Type	integer

Content/value	Values have the following meanings: 0 Prefix, no separator 1 Suffix, no separator 2 Prefix, 1 character separator 3 Suffix, 1 character separator.
Application name	"PM_National"
Key name	"iDigits"
Type	integer
Content/value	n = number of decimal digits.
Application name	"PM_National"
Key name	"iTime"
Type	integer
Content/value	0 = 12-hour clock; 1 = 24-hour clock.
Application name	"PM_National"
Key name	"iLzero"
Type	integer
Content/value	0 = no leading zero; 1 = leading zero.
Application name	"PM_National"
Key name	"s1159"
Type	string
Content/value	"am" for example. 3 chars max.
Application name	"PM_National"
Key name	"s2359"
Type	string
Content/value	"pm" for example. 3 chars max.
Application name	"PM_National"
Key name	"sCurrency"
Type	string
Content/value	"$" for example. 3 chars max.
Application name	"PM_National"
Key name	"sThousand"
Type	string
Content/value	"," for example. 1 char max.
Application name	"PM_National"
Key name	"sDecimal"
Type	string
Content/value	"." for example. 1 char max.
Application name	"PM_National"
Key name	"sDate"
Type	string
Content/value	"/" for example. 1 char max.
Application name	"PM_National"
Key name	"sTime"
Type	string
Content/value	":" for example. 1 char max.
Application name	"PM_National"
Key name	"sList"
Type	string
Content/value	"," for example. 1 char max.
Application name	"PM_Fonts"
Key name	
Type	string
Content/value	fully-qualified drive:\path\filename.ext.
Application name	"PM_SPOOLER"
Key name	"QUEUE"
Type	string

Content/value <Queue name>;

where:

- <Queue name> is the name of the default queue (might be NULL). This must be a key name for the PM_SPOOLER_QUEUE application.

Application name "PM_SPOOLER"
Key name "PRINTER"
Type string
Content/value <Printer name>;

where:

- <Printer name> is the name of the default printer (might be NULL).

Application name "PM_SPOOLER_PRINTER"
Key name <Printer name>
Type string
Content/value <Addr>;<DDL>;<QL>;<NetOpt>;

where:

- <Addr> is a (possibly null) character string, that gives the logical address of the printer.
- <DDL> is a (possibly null) character string defined as follows:

```
<DDL>      = (<DDEntry>,)* <DDEntry>
           | NULL
<DDEntry>  = <DDName>
           | <DDName>.<DevName>
```

 where:
 - <DDName> (character string) is the name of a device driver.
 - <DevName> (character string) is the name of a device type.
- <QL> is a (possibly null) character string defined as follows:

```
<QL>       = (<QName>,)* <QName>
           | NULL
```

 where:
 - <QName> (character string) is a queue name.
- <NetOpt> is a (possibly null) character string that gives network options.

Note: The character "*" means that the preceding field can be present any number of times or not at all.

Application name "PM_SPOOLER_QUEUE_DESCR"
Key name <Queue name>
Type string
Content/value <Queue description>;

where:

- <Queue description> is a (possibly NULL) character string specifying a description of this queue (maximum length 48 characters excluding the zero). This string can contain any symbol except the semicolon (;).

Application name "PM_SPOOLER_QUEUE_DD"
Key name <Queue name>
Type string
Content/value <Device driver name>;

where:

- <Device driver name> is a (possibly NULL) character string that specifies the default device driver for the queue.

Application name "PM_SPOOLER_QUEUE_DDDATA"
Key name <Queue name>
Type binary data
Content/value <Device driver data>;

where:

- <Device driver data> is binary data as returned by DevPostDeviceModes (specifying DPDM_POSTJOBPROP), for the default device driver for the queue.

Application name	"PM_SPOOLER_PRINTER_DESCR"
Key name	<Printer name>
Type	binary data
Content/value	<Printer description>;

where:

- <Printer description> is a (possibly NULL) character string specifying a description of this printer (maximum length 48 characters excluding the zero). This string can contain any symbol except the semicolon (;).

Appendix F. Virtual Key Definitions

The PC VKEY set is shown in the following table:

Symbol	Personal Computer AT Keyboard	Enhanced Keyboard
VK_BUTTON1 VK_BUTTON2 VK_BUTTON3	These values are only used to access the up/down and toggled states of the pointing device buttons; they never actually appear in a WM_CHAR message.	
VK_BREAK	Ctrl + Scroll Lock	Ctrl + Pause
VK_BACKSPACE	Backspace	Backspace
VK_TAB	Tab	Tab
VK_BACKTAB	Shift + Tab	Shift + Tab
VK_NEWLINE	New Line	New Line
VK_SHIFT *	Left and Right Shift	Left and Right Shift
VK_CTRL *	Ctrl	Left and Right Ctrl
VK_ALT *	Alt	Left and Right Alt
VK_ALTGRAF *	None	Alt Graf (if available)
VK_PAUSE	Ctrl + Num Lock	Pause
VK_CAPSLOCK	Caps Lock	Caps Lock
VK_ESC	Esc	Esc
VK_SPACE *	Space	Space
VK_PAGEUP *	Numpad 9	Pg Up and Numpad 9
VK_PAGEDOWN *	Numpad 3	Pg Dn and Numpad 3
VK_END *	Numpad 1	End and Numpad 1
VK_HOME *	Numpad 7	Home and Numpad 7
VK_LEFT *	Numpad 4	Left and Numpad 4
VK_UP *	Numpad 8	Up and Numpad 8
VK_RIGHT *	Numpad 6	Right and Numpad 6
VK_DOWN *	Numpad 2	Down and Numpad 2
VK_PRINTSCRN	Shift + Print Screen	Print Screen
VK_INSERT *	Numpad 0	Ins and Numpad 0
VK_DELETE *	Numpad .	Del and Numpad .
VK_SCRLLOCK	Scroll Lock	Scroll Lock
VK_NUMLOCK	Num Lock	Num Lock
VK_ENTER	Shift + New Line	Shift + New Line and Numpad Enter
VK_SYSRQ	SysRq	Alt + Print Screen
VK_F1 *	F1	F1
VK_F2 *	F2	F2
VK_F3 *	F3	F3
VK_F4 *	F4	F4

Symbol	Personal Computer AT Keyboard	Enhanced Keyboard
VK_F5 *	F5	F5
VK_F6 *	F6	F6
VK_F7 *	F7	F7
VK_F8 *	F8	F8
VK_F9 *	F9	F9
VK_F10 *	F10	F10
VK_F11 *	None	F11
VK_F12 *	None	F12
VK_F13	None	None
VK_F14	None	None
VK_F15	None	None
VK_F16	None	None
VK_F17	None	None
VK_F18	None	None
VK_F19	None	None
VK_F20	None	None
VK_F21	None	None
VK_F22	None	None
VK_F23	None	None
VK_F24	None	None

Notes:

1. VKEYs marked with an asterisk (*) are generated irrespective of other shift states (Shift, Ctrl, Alt, and Alt Graf).
2. VK_CAPSLOCK is **not** generated for any of the Ctrl shift states, for PC-DOS compatibility.
3. Wherever possible, the VK_ name is derived from the legend on the key top of the 101-key Enhanced PC keyboard.

Glossary

This glossary defines terms used in this book. It contains terms specific to the Presentation Manager, but it is not a complete glossary for OS/2 Version 1.2. This glossary includes terms and definitions from the *IBM Dictionary of Computing*, SC20-1699.

A

accelerator. A single key stroke that invokes an application-defined function.

accelerator table. Used to define which key strokes are treated as **accelerators** and the commands they are translated into.

action bar. The area at the top of a window that contains the choices currently available in the application program.

action point. The current position on the screen at which the pointer is pointing. (Contrast with **hot spot** and **input focus**.)

active program. A program currently running on the computer. See also **interactive program, noninteractive program**, and **foreground program**.

active window. The window with which the user is currently interacting.

alphanumeric video output. Output to the logical video buffer when the video adapter is in text mode and the logical video buffer is addressed by an application as a rectangular array of character cells.

anchor block. An area of Presentation Manager-internal resources allocated to a process or thread that calls WinInitialize.

anchor point. A point in a window used by a program designer or by a window manager to position a subsequently appearing window.

ANSI. American National Standards Institute.

APA. All points addressable.

API. Application programming interface. The formally-defined programming language that is between an IBM application program and the user of the program. See also **GPI**.

area. In computer graphics, a filled shape such as a solid rectangle.

ASCII. American National Standard Code for Information Interchange.

ASCIIZ. A string of ASCII characters that is terminated with a byte containing the value 0.

aspect ratio. In computer graphics, the width-to-height ratio of an area, symbol, or shape.

asynchronous. (1) Without regular time relationship. (2) Unexpected or unpredictable with respect to the execution of a program's instructions.

atom. A constant that represents a string. Once a string has been defined as an atom, the atom can be used in place of the string to save space. Strings are associated with their respective atoms in an **atom table**. See also **integer atom**.

atom table. Used to relate **atoms** with the strings that they represent. Also in the table is the mechanism by which the presence of a string can be checked.

attributes. Characteristics or properties that can be controlled, usually to obtain a required appearance; for example, the color of a line. See also **graphics attributes** and **segment attributes**.

AVIO. Advanced Video Input/Output.

B

background color. The color in which the background of a graphic primitive is drawn.

background mix. An attribute that determines how the background of a graphic primitive is combined with the existing color of the graphics presentation space. Contrast with **mix**.

Bézier curves. A mathematical technique of specifying smooth continuous lines and surfaces, which require a starting point and a finishing point with several intermediate points that influence or control the path of the linking curve. Named after Dr. P. Bézier.

bit map. A representation in memory of the data displayed on an APA device, usually the screen.

border. A visual indication (for example, a separator line or a background color) of the boundaries of a window.

bucket. One or more fields in which the result of an operation is kept.

button. A mechanism on a **pointing device**, such as a mouse, used to request or initiate an action. Contrast with **pushbutton** and **radio button**.

C

cached micro presentation space. A presentation space from a Presentation Manager-owned store of micro presentation spaces. It can be used for drawing to a window only, and must be returned to the store when the task is complete.

cancel. An action that removes the current window or menu without processing it, and returns the previous window.

cell. See **character cell**.

CGA. Color graphics adapter.

chained list. A list in which the data elements may be dispersed but in which each data element contains information for locating the next. Synonym for **linked list.**

character. A letter, digit, or other symbol.

character box. In computer graphics, the boundary that defines, in world coordinates, the horizontal and vertical space occupied by a single character from a character set. See also **character mode.** Contrast with **character cell.**

character cell. The physical, rectangular space in which any single character is displayed on a screen or printer device. Position is addressed by row and column coordinates. Contrast with **character box.**

character code. The means of addressing a character in a character set, sometimes called **code point.**

character mode. The character mode, in conjunction with the font type, determines the extent to which graphics characters are affected by the character box, shear, and angle attributes.

check box. A control window, shaped like a square button on the screen, that can be in a checked or unchecked state. It is used to select one or more items from a list. Contrast with **radio button.**

check mark. The symbol (√) that is used to indicate a selected item on a pull-down.

child window. A window that is positioned relative to another window (either a main window or another child window). Contrast with **parent window.**

choice. An option that can be selected. The choice can be presented as text, as a symbol (number or letter), or as an icon (a pictorial symbol).

class. See **window class.**

class style. The set of properties that apply to every window in a window class.

client area. The area in the center of a window that contains the main information of the window.

clipboard. An area of main storage that can hold data being passed from one Presentation Manager application to another. Various data formats can be stored.

clipping. In computer graphics, removing those parts of a display image that lie outside a given boundary.

clip limits. The area of the paper that can be reached by a printer or plotter.

clipping path. A clipping boundary in world-coordinate space.

code page. An assignment of graphic characters and control-function meanings to all code points.

code point. Synonym for **character code.**

color dithering. See **dithering**

command line. On a display screen, a display line usually at the bottom of the screen, in which only commands can be entered.

Common Programming Interface. A consistent set of specifications for languages, commands, and calls to enable applications to be developed across all SAA environments. See also **Systems Application Architecture.**

Common User Access. A set of rules that define the way information is presented on the screen, and the techniques for the user to interact with the information.

control. The means by which an operator gives input to an application. A **choice** corresponds to a control.

Control Panel. In the Presentation Manager, a program used to set up user preferences that act globally across the system.

Control Program. The basic function of OS/2 including DOS emulation and the support for keyboard, mouse, and VIO.

control window. A class of window used to handle a specific kind of user interaction. Radio buttons and check boxes are examples.

correlation. The action of determining which element or object within a picture is at a given position on the display. This follows a **pick** operation.

CPI. Common Programming Interface.

current position. The point from which the next primitive will be drawn.

cursor. A symbol displayed on the screen and associated with an input device. The cursor indicates where input from the device will be placed. Types of cursors include text cursors, graphics cursors, and selection cursors. Contrast with **pointer** and **input focus.**

D

data structure. (ISO) The syntactic structure of symbolic expressions and their storage-allocation characteristics.

DBCS. See **double-byte character set.**

decipoint. In printing, one tenth of a point. There are 72 points in an inch.

default procedure. Function provided by the Presentation Interface that may be used to process standard messages from dialogs or windows.

default value. A value used when no value is explicitly specified by the user. For example, in the graphics programming interface, the default line-type is 'solid'.

Desktop Manager. In the Presentation Manager, a window which displays a list of groups of programs, each of which can be started or stopped.

desktop window. The window, corresponding to the physical device, against which all other types of windows are established.

device context. A logical description of a data destination such as memory, metafile, display, printer, or plotter. See also **direct device context, information device context, memory device context, metafile device context, queued device context,** and **screen device context.**

device driver. A file that contains the code needed to attach and use a device such as a display, printer, or plotter.

device space. Coordinate space in which graphics are assembled after all GPI transformations have been applied. Device space is defined in device-specific units.

dialog. The interchange of information between a computer and its user through a sequence of requests by the user and the presentation of responses by the computer.

dialog box. A type of window that contains one or more controls for the formatted display and entry of data. Also known as a **pop-up window**. A modal dialog box is used to implement a pop-up window.

Dialog Box Editor. A **WYSIWYG** editor that creates dialog boxes for communicating with the application user.

dialog item. A component (for example, a menu or a button) of a dialog box. Dialog items are also used when creating dialog templates.

dialog template. The definition of a dialog box, which contains details of its position, appearance, and window ID, and the window ID of each of its child windows.

direct device context. A logical description of a data destination that is a device other than the screen (for example, a printer or plotter), and where the output is not to go through the spooler. Its purpose is to satisfy queries. See also **device context.**

direct manipulation. The action of using the mouse to move objects around the screen. For example, moving files and directories about in the **File Manager.**

directory. A type of file containing the names and controlling information for other files or other directories.

display point. Synonym for **pel.**

dithering. The process used in color displays whereby every other pel is set to one color, and the intermediate pels are set to another. Together they produce the effect of a third color at normal viewing distances. This process can only be used on solid areas of color; it does not work on narrow lines, for example.

double-byte character set (DBCS). A set of characters in which each character is represented by two bytes. Languages such as Japanese, Chinese, and Korean, which contain more characters than can be represented by 256 code points, require double-byte character sets. As each character requires two bytes, the entering, displaying, and printing of DBCS characters requires hardware and software that can support DBCS.

dragging. In computer graphics, moving an object on the display screen as if it were attached to the pointer.

drawing chain. See **segment chain.**

drop. To fix the position of an object that is being dragged, by releasing the select button of the pointing device.

dynamic segments. Graphics segments drawn in exclusive-OR mix mode so that they can be moved from one screen position to another without affecting the rest of the displayed picture.

E

EBCDIC. Extended binary-coded decimal interchange code.

EGA. Extended graphics adapter.

element. An entry in a graphics segment that comprises one or more graphics orders and that is addressed by the element pointer.

entry field. An area on the screen, usually highlighted in some manner, in which users type information.

entry-field control. The means by which the application receives data entered by the user in an entry field. When it has the input focus, it displays a flashing pointer at the position where the next typed character will go.

exit. The action that terminates the current function and returns the user to a higher level function. Repeated exit requests return the user to the point from which all functions provided to the system are accessible. Contrast with **cancel.**

extended-choice selection. A mode that allows the user to select more than one item from a window. Not all windows allow extended choice selection. Contrast with **multiple-choice selection.**

F

File Manager. In the Presentation Manager, a program that displays directories and files, and allows various actions on them.

file specification. The full identifier for a file, which includes its file name, extension, path, and drive.

fillet. A curve that is tangential to the end points of two adjoining lines. See also **polyfillet.**

focus. See **input focus.**

font. A particular size and style of typeface that contains definitions of character sets, marker sets, and pattern sets.

foreground program. The program with which the user is currently interacting. Also known as **interactive program.**

frame. The part of a window that can contain several different visual elements specified by the application, but drawn and controlled by the Presentation Manager. The frame encloses the client area.

frame styles. Different standard window layouts provided by the Presentation Manager.

full-screen application. An application program that occupies the whole screen.

function key area. The area at the bottom of a window that contains function key assignments such as F1=Help.

G

Global Descriptor Table (GDT). Defines code and data segments available to all tasks in an application.

glyph. A graphic symbol whose appearance conveys information.

GPI. Graphics programming interface. The formally-defined programming language that is between an IBM graphics program and the user of the program. See also **API.**

graphics. A picture defined in terms of graphic primitives and graphics attributes.

graphics attributes. Attributes that apply to graphic primitives. Examples are color, line type, and shading-pattern definition. See also **segment attributes.**

graphics field. The clipping boundary that defines the visible part of the presentation-page contents.

graphics model space. The conceptual coordinate space in which a picture is constructed after any model transforms have been applied. Also known as **model space.**

graphic primitive. A single item of drawn graphics, such as a line, arc, or graphics text string. See also **graphics segment.**

graphics segment. A sequence of related graphic primitives and graphics attributes. See also **graphic primitive.**

graying. The indication that a choice on a pull-down is unavailable.

group. A collection of logically-connected controls. For example, the buttons controlling paper size for a printer. See also **program group.**

H

handle. An identifier that represents an object, such as a device or window, to the Presentation Interface.

hard error. An error condition on a network that requires either that the system be reconfigured, or that the source of the error be removed before the system can resume reliable operation.

heap. An area of free storage available for dynamic allocation by an application. Its size varies according to the storage requirements of the application.

hit testing. The means of identifying which window is associated with which input device event.

hook. A mechanism by which procedures are called when certain events occur in the system. For example, the filtering of mouse and keyboard input before it is received by an application program.

hook chain. A sequence of hook procedures that are "chained" together so that each event is passed, in turn, to each procedure in the chain.

hot spot. The part of the pointer that must touch an object before it can be selected. This is usually the tip of the pointer. Contrast with **action point.**

I

icon. A pictorial representation of an item the user can select. Icons can represent items (such as a document file) that the user wants to work on, and actions that the user wants to perform. In the Presentation Manager, icons are used for data objects, system actions, and minimized programs.

icon area. In the Presentation Manager, the area at the bottom of the screen that is normally used to display the icons for minimized programs.

Icon Editor. The Presentation Manager-provided tool for creating icons.

image font. A set of symbols, each of which is described in a rectangular array of pels. Some of the pels in the array are set to produce the image of the symbol. Contrast with **outline font.**

information device context. A logical description of a data destination other than the screen (for example, a printer or plotter), but where no output will occur. Its purpose is to satisfy queries. See also **device context.**

input focus. The area of the screen that will receive input from an input device (typically the keyboard).

input router. OS/2-internal process that removes messages from the system queue.

interactive graphics. Graphics that can be moved or manipulated by a user at a terminal.

integer atom. A special kind of **atom** that represents a predefined system constant and carries no storage overhead. For example, names of window classes provided by Presentation Manager are expressed as integer atoms.

interactive program. A program that is running (active) and is ready to receive (or is receiving) input from the user. Compare with **active program** and contrast with **noninteractive program.**

Also known as a **foreground program.**

interchange file. Data that can be sent from one Presentation Interface application to another.

J

journal. A special-purpose file that is used to record changes made in the system.

K

Kanji. A graphic character set used in Japanese ideographic alphabets.

kerning. The design of graphics characters so that their character boxes overlap. Used to space text proportionally.

L

label. In a graphics segment, an identifier of one or more elements that is used when editing the segment.

language support procedure. Function provided by the Presentation Interface for applications that do not, or cannot (as in the case of COBOL and FORTRAN programs), provide their own dialog or window procedures.

LIFO stack. A data stack from which data is retrieved in last-in, first-out order.

linked list. Synonym for **chained list.**

list box. A control window containing a vertical list of selectable descriptions.

Local Descriptor Table (LDT). Defines code and data segments specific to a single task.

M

main window. The window that is positioned relative to the **desktop window.**

marker box. In computer graphics, the boundary that defines, in world coordinates, the horizontal and vertical space occupied by a single marker from a marker set.

marker symbol. A symbol centered on a point. Graphs and charts can use marker symbols to indicate the plotted points.

maximize. A window-sizing action that makes the window the largest size possible.

media window. The part of the physical device (display, printer, or plotter) on which a picture is presented.

memory device context. A logical description of a data destination that is a memory bit map. See also **device context.**

menu. A type of panel that consists of one or more selection fields. Also called a **menu panel.**

message. A packet of data used for communication between the Presentation Interface and windowed applications.

message filter. The means of selecting which messages from a specific window will be handled by the application.

message queue. A sequenced collection of messages to be read by the application.

metafile. The generic name for the definition of the contents of a picture. Metafiles are used to allow pictures to be used by other applications.

metafile device context. A logical description of a data destination that is a metafile, which is used for graphics interchange. See also **device context.**

metalanguage. A language used to specify another language. In this publication, the data types are described using a metalanguage so as to make the descriptions independent of any one computer language.

micro presentation space. A graphics presentation space in which a restricted set of the GPI function calls is available.

minimize. A window-sizing action that makes the window the smallest size possible. In the Presentation Manager, minimized windows are represented by icons.

mix. An attribute that determines how the foreground of a graphic primitive is combined with the existing color of graphics output. Also known as **foreground mix.** Contrast with **background mix.**

mixed character string. A string containing a mixture of one-byte and **Kanji** or Hangeul (two-byte) characters.

mnemonic. A method of selecting an item on a pull-down by means of typing the highlighted letter in the menu item.

modal dialog box. The type of control that allows the operator to perform input operations on only the current dialog box or one of its child windows. Also known as a **serial dialog box.** Contrast with **parallel dialog box.**

modeless dialog box. The type of control that allows the operator to perform input operations on any of the application's windows. Also known as a **parallel dialog box.** Contrast with **modal dialog box.**

model space. See **graphics model space.**

mouse. A hand-held device that is moved around to position the pointer on the screen.

multiple-choice selection. A mode that allows users to select any number of choices, including none at all. See also **check box.** Contrast with **extended-choice selection.**

multitasking. The concurrent processing of applications or parts of applications. A running application and its data are protected from other concurrently running applications.

N

named pipe. A named object that provides client-to-server, server-to-client, or full duplex communication between unrelated processes. Contrast with **unnamed pipe.**

noninteractive program. A program that is running (active) but is not ready to receive input from the user. Compare with **active program,** and contrast with **interactive program.**

nonretained graphics. Graphic primitives that are not remembered by the Presentation Interface once they have been drawn. Contrast with **retained graphics.**

null-terminated string. A string of (n + 1) characters where the (n + 1)th character is the 'null' character (X'00'), and is used to represent an n-character string with implicit length. Also known as 'zero-terminated' string and 'ASCIIZ' string.

O

object window. A window that does not have a parent, but which may have child windows. An object window cannot be presented on a device.

open. To start working with a file, directory, or other object.

outline font. A set of symbols, each of which is created as a series of lines and curves. Contrast with **image font.**

output area. The area of the output device within which the picture is to be displayed, printed, or plotted.

owner window. A window into which specific events that occur in another (owned) window are reported.

P

page viewport. A boundary in device coordinates that defines the area of the output device in which graphics are to be displayed. The presentation-page contents are transformed automatically to the page viewport in device space.

paint. The action of drawing or redrawing the contents of a window.

panel. A particular arrangement of information grouped together for presentation to the user in a window.

panel body area. The part of a window not occupied by the action bar or function key area. The panel body area may contain information, selection fields, and entry fields. Also known as **client area.**

panel body area separator. A line or color boundary that provides users with a visual distinction between two adjacent areas of a panel.

panel title. A panel element that identifies the information in the panel.

paper size. The size of paper, defined in either standard U.S. or European names (for example, A, B, A4), and measured in inches or millimeters respectively.

parallel dialog box. See **modeless dialog box.**

parent window. The window relative to which one or more child windows are positioned. Contrast with **child window.**

pel. The smallest area of a display screen capable of being addressed and switched between visible and invisible states. Synonym for **display point, pixel,** and **picture element.**

pick. To select part of a displayed object using the pointer.

picture chain. See **segment chain.**

picture element. Synonym for **pel.**

pipe. See **named pipe, unnamed pipe.**

pixel. Synonym for **pel.**

plotter. An output device that uses pens to draw its output on paper or transparency foils.

pointer. The symbol displayed on the screen that is moved by a pointing device, such as a **mouse.** The pointer is used to point at items that users can select. Contrast with **cursor.**

pointing device. A device (such as a mouse) used to move a pointer on the screen.

pointings. Pairs of x-y coordinates produced by an operator defining positions on a screen with a pointing device, such as a **mouse.**

polyfillet. A curve based on a sequence of lines. It is tangential to the end points of the first and last lines, and tangential also to the midpoints of all other lines. See also **fillet.**

polyline. A sequence of adjoining lines.

pop. To retrieve an item from a last-in-first-out stack of items. Contrast with **push.**

pop-up window. A window that appears on top of another window in a dialog. Each pop-up window must be completed before returning to the underlying window.

Presentation Manager. The visual component of OS/2 that presents, in windows, a graphics-based interface to applications and files installed and running in OS/2.

presentation page. The coordinate space in which a picture is assembled for display.

presentation space (PS). Contains the device-independent definition of a picture.

primary window. The window in which the main dialog between the user and the program takes place. See also **secondary window.**

primitive. See **graphic primitive.**

primitive attribute. A specifiable characteristic of a graphic primitive. See **graphics attributes.**

print job. The result of sending a document or picture to be printed.

Print Manager. In the Presentation Manager, the part of the spooler that manages the spooling process. It also allows users to view print queues and to manipulate print jobs.

process. An instance of an executing application and the resources it is using.

program details. Information about a program that is specified in the **Program Manager** window and is used when the program is started.

program group. In the Presentation Manager, several programs that can be acted upon as a single entity.

program name. The full file specification of a program. Contrast with **program title.**

program title. The name of a program as it is listed in the **Program Manager** window. Contrast with **program name.**

pull-down. An **action bar** extension that displays a list of choices available for a selected action bar choice.

push. To add an item to a last-in-first-out stack of items. Contrast with **pop.**

pushbutton. A control window, shaped like a round-cornered rectangle and containing text, that invokes an immediate action, such as 'enter' or 'cancel'.

Q

queue. A list of print jobs waiting to be printed.

queued device context. A logical description of a data destination (for example, a printer or plotter) where the output is to go through the spooler. See also **device context.**

R

radio button. A control window, shaped like a round button on the screen, that can be in a checked or unchecked state. It is used to select a single item from list. Contrast with **check box.**

realize. To cause the system to ensure, wherever possible, that the physical color table of a device is set to the closest possible match in the logical color table.

reentrant. The attribute of a program or routine that allows the same copy of the program or routine to be used concurrently by two or more tasks.

refresh. To update a window, with changed information, to its current status.

region. A clipping boundary in device space.

resource. The means of providing extra information used in the definition of a window. A resource can contain definitions of fonts, templates, accelerators, and mnemonics; the definitions are held in a resource file.

restore. To return a window to its original size or position following a sizing or moving action.

retained graphics. Graphic primitives that are remembered by the Presentation Interface after they have been drawn. Contrast with **nonretained graphics.**

reverse video. A form of alphanumeric highlighting for a character, field, or cursor, in which its color is exchanged with that of its background. For example, changing a red character on a black background to a black character on a red background.

RGB. Red-green-blue. For example "RGB display."

roman. Relating to a type style, with upright characters.

S

SAA. Systems Application Architecture.

screen. The physical surface of a work station or terminal upon which information is presented to users.

screen device context. A logical description of a data destination that is a particular window on the screen. See also **device context.**

scroll bar. A control window, horizontally or vertically aligned, that allows the user to scroll additional data into an associated panel area.

scrollable entry field. An entry field larger than the visible field.

scrollable selection field. A selection field that contains more choices than are visible.

scrolling. Moving a display image vertically or horizontally in a manner such that new data appears at one edge, as existing data disappears at the opposite edge.

secondary window. A type of window associated with the primary window in a dialog. A secondary window begins a secondary and parallel dialog that runs at the same time as the primary dialog.

segment. See **graphics segment.**

segment attributes. Attributes that apply to the segment as an entity, as opposed to the individual primitives within the segment. For example, the visibility or detectability of a segment.

segment chain. All segments in a graphics presentation space that are defined with the 'chained' attribute. Synonym for **picture chain.**

segment priority. The order in which segments are drawn.

segment store. An area in a normal graphics presentation space where retained graphics segments are stored.

select. To mark or choose an item. Note that **select** means to mark or type in a choice on the screen; **enter** means to send all selected choices to the computer for processing.

select button. The button on a pointing device, such as a mouse, that is pressed to select a menu choice. Also known as button 1.

selection cursor. A type of cursor used to indicate the choice or entry field users want to interact with. It is represented by highlighting the item it is currently positioned on.

selection field. A field containing a list of choices from which the user can select one or more.

semaphore. An object used by multi-threaded applications for signaling purposes and for controlling access to serially reusable resources.

separator. See **panel body area separator.**

serial dialog box. See **modal dialog box.**

serially reusable resource (SRR). A logical resource or object that can be accessed by only one task at a time.

session. A routing mechanism for user interaction via the console.

shadow box. The area on the screen that follows mouse movements and shows what shape the window will take if the mouse button is released.

shear. The tilt of graphics text when each character leans to the left or right while retaining a horizontal baseline.

Shutdown. The procedure required before the computer is switched off to ensure that data is not lost.

sibling windows. Child windows that have the same parent window.

slider box. An area on the scroll bar that indicates the size and position of the visible information in a panel area in relation to the information available. Also known as **thumb mark.**

spline. A sequence of one or more Bézier curves.

spooler. A program that intercepts the data going to printer devices and writes it to disk. The data is printed or plotted when it is complete, and the required device is available. The spooler prevents output from different sources being intermixed.

standard window. A collection of windows that form a panel.

static control. The means by which the application presents descriptive information (for example, headings and descriptors) to the user. The user cannot change this information.

style. See **window style.**

suballocation. The allocation of a part of one extent for occupancy by elements of a component other than the one occupying the remainder of the extent.

switch. An action that moves the input focus from one area to another. This can be within the same window or from one window to another.

switch list. See **Task List**

symbolic identifier. A text string that equates to an integer value in an include file, that is used to identify a programming object.

System Menu. In the Presentation Manager, the pull-down in the top left corner of a window that allows it to be moved and sized with the keyboard.

system queue. This is the master queue for all pointer device or keyboard events.

Systems Application Architecture. A formal set of rules that enables applications to be run without modification in different computer environments.

T

tag. One or more characters attached to a set of data that defines the formatting or other characteristics of the set, including its definition.

Task list. In the Presentation Manager, the list of programs that are active. The list can be used to switch to a program and to stop programs.

template. An ASCII-text definition of an action bar and pull-down menu, held in a resource file, or as a data structure in program memory.

text. Characters or symbols.

text cursor. A symbol displayed in an entry field that indicates where typed input will appear.

text window. Also known as the VIO window.

thread. A unit of execution within a process.

thumb mark. The portion of the scroll bar that describes the range and properties of the data that is currently visible in a window. Also known as a **slider box.**

tilde. A mark used to denote the character that is to be used as a mnemonic when selecting text items within a menu.

title bar. The area at the top of a window that contains the window title. The title bar is highlighted when that window has the input focus. Contrast with **panel title.**

transform. (1) The action of modifying a picture by scaling, shearing, reflecting, rotating, or translating. (2) The object that performs or defines such a modification; also referred to as a **transformation.**

Tree. In the Presentation Manager, the window in the **File Manager** that shows the organization of drives and directories.

U

unnamed pipe. A circular buffer, created in memory, used by related processes to communicate with one another. Contrast with **named pipe.**

update region. A system-provided area of dynamic storage containing one or more (not necessarily contiguous) rectangular areas of a window, that are visually invalid or incorrect, and therefore in need of repainting.

User Shell. A component of OS/2 that uses a graphics-based, windowed interface to allow the user to manage applications and files installed and running under OS/2.

V

VGA. Video graphics array.

viewing pipeline. The series of transformations applied to a graphic object to map the object to the device on which it is to be presented.

viewing window. Clipping boundary that defines the visible part of model space.

VIO. Video Input/Output.

virtual memory (VM). Addressable space that is apparent to the user as the processor storage space, but not having a fixed physical location.

visible region. A window's presentation space, clipped to the boundary of the window and the boundaries of any overlying window.

W

wild-card character. The global file-name characters ? or *.

window. A rectangular area of the screen with visible boundaries in which information is displayed. A window may be smaller than or equal to the screen. Windows may overlap on the screen and give the appearance of one window being on top of another.

window class. The grouping of windows whose processing needs conform to the services provided by one window procedure.

window coordinates. The means by which a window position or size is defined; measured in device units, or **pels.**

window procedure. Code that is activated in response to a message.

window rectangle. The means by which the size and position of a window is described in relation to the desktop window.

window style. The set of properties that influence how events related to a particular window will be processed.

workstation. A display screen together with attachments such as a keyboard, a local copy device, or a tablet.

world coordinates. Application-convenient coordinates used for drawing graphics.

world-coordinate space. Coordinate space in which graphics are defined before transformations are applied.

WYSIWYG. What You See Is What You Get. A capability that enables text to be displayed on a screen in the same way it will be formatted on a printer.

Z

z-order. The order in which sibling windows are presented. The topmost sibling window obscures any portion of the siblings that it overlaps; the same effect occurs down through the order of lower sibling windows.

zooming. In graphics applications, the process of increasing or decreasing the size of picture.

Index

A

ABB_* values 4-238, 4-273
ACCEL 2-1
accelerator table
 copy 9-27
 create 9-31
 destroy 9-65
 load 9-135
 query 9-170
 set 9-267
 translate 9-326
ACCELTABLE 2-1
ACCELTABLE statement 26-8
Add Atom 9-10
Add Program 6-2, 9-11
Add Switch Entry 9-12
alarm sound 9-13
Allocate Heap Space 9-14
AM_* values 4-130, 4-235
application-supplied functions 10-1
Applications
 full screen VIO 28-1
 Windowed PM 28-2
Arabic text 4-256
arc
 create 4-115
 full 4-85, 4-109
 partial 4-108
 query parameters 4-129
 set current parameters 4-233
 set default parameters 4-271
Arc at a Given Position 27-3
Arc at Current Position 27-3
ARCPARAM 2-1
AREABUNDLE 2-2
areas
 begin construction 4-11
 construction of interior 4-13
 end construction 4-77
arrays
 convert 4-39
ASCII 9-192, 9-278, 28-18
ASCII MIXED code pages 28-18
Associate 4-10
Associate Help Instance 9-15
ASSOCTABLE statement 26-10
ATOM 2-2
attribute primitive type 4-237
attribute primitive types 4-272
attribute values
 character 4-237, 4-272
 image 4-238, 4-273
 line 4-237, 4-272
 marker 4-238, 4-272
 pattern (area) 4-238, 4-272
attributes
 character-set 4-260
 color 4-267
 cosmetic line width 4-295
 foreground color mix 4-305
 geometric line width 4-296
 line type 4-293
attributes *(continued)*
 line width 4-295
 marker box 4-300
 marker set 4-302
 marker symbol 4-298
 pattern 4-310
 pattern set 4-313
 query mode 4-130
 restore saved 4-124
 segment 4-322
 set 4-237
 set default 4-272
 set line-end 4-289
 set line-join 4-291
 specify mode 4-235
ATTR_* values 4-175, 4-205, 4-287, 4-321
Average character width C-4

B

background
 query color 4-133, 4-134
 query color-mixing mode 4-134
 query mix 4-134
BANDRECT 2-2
BA_* values 4-11
BBO_* values 4-20, 4-340
BDS_* values 13-3
Begin Area 4-11, 27-3
Begin Element 4-14, 27-4
Begin Image at Current Position 27-4
Begin Image at Given Position 27-4
Begin Paint 9-18
Begin Path 4-16, 27-5
Begin Window Enumeration 9-17
Bézier Curve at Current Position 27-6
Bézier Curve at Given Position 27-6
Bézier splines, create 4-122
bindings references v
Bit Blt 4-19
bit maps
 color 4-21, 4-341
 copy rectangle of image data 4-19, 4-339
 create 4-48
 data B-1
 delete 4-60
 draw 9-79
 example B-1
 file format B-2
 get system 9-118
 information tables B-1
 load 4-94
 monochrome 4-21, 4-341
 query bits 4-135
 query device formats 4-163
 query dimension 4-136
 query handle 4-137
 query number of local identifiers 4-191
 query parameters 4-138
 query set identifiers 4-209
 set as currently selected 4-248
 set bits 4-249

bit maps *(continued)*
set identifier 4-251
standard formats B-1
transfer data from application storage 4-249
bit-map tag
delete 4-67
Bitblt 27-6
BITMAPINFO 2-2
BITMAPINFOHEADER 2-3
bits
modify an integer under control of a mask 9-270
modify in an integer 9-269
query in integer 9-176
query in integer under mask 9-177
set 9-269
set under mask 9-270
BIT1 2-3
BIT16 2-3
BIT32 2-3
BIT4 2-3
BIT6 2-3
BIT8 2-3
BMSG_* values 9-19
BM_CLICK 13-5
BM_QUERYCHECK 13-5
BM_QUERYCHECKINDEX 13-6
BM_QUERYHILITE 13-7
BM_SETCHECK 13-7
BM_SETDEFAULT 13-8
BM_SETHILITE 13-9
BM_* values 4-134, 4-246
BN_* values 13-3
BOOL 2-3
Box 4-23
draw 4-23
Box at Current Position 27-7
Box at Given Position 27-7
Broadcast Message 9-19
BS_* values 13-1
BTNCDATA 13-2
BUFFER 2-3
BUNDLE 2-3
button control data 13-2
button control styles 13-1
button control window processing 13-1
button filtering constants 9-112
BYTE 2-3

C

C language v
Calculate Frame Rectangle 9-20
Call Message Filter 9-21
Call Segment 27-8
Call Segment Matrix 4-25
Cancel Shutdown 9-22
CAPS_* values 3-14
CATCHBUF 2-3
CBB_* values 4-237, 4-272
CBM_HILITE 19-4
CBM_ISLISTSHOWING 19-4
CBM_SHOWLIST 19-5
CBM_* values 4-48
CBN_* values 19-2
CBS_* values 19-1
CFI_* values 9-182, 9-273
CF_* values 9-273
chain
draw 4-70
chained attribute for segments
modify (GpiSetSegmentAttrs) 4-322
Change Focus Window 9-99
Change Program 6-3
Change Switch Entry 9-24
CHAR 2-3
character
convert to uppercase 9-330
query angle 4-140
query box 4-141
query direction 4-142
query mode 4-143
query set 4-144
query shear 4-145
query string positions 4-146
query string positions at 4-147
set angle 4-252
set box 4-254
set direction 4-256
set mode 4-258
set set 4-260
set shear 4-261
character attribute values 4-237, 4-272
character definitions
font C-8
character direction
Arabic text 4-256
Chinese text 4-256
Roman text 4-256
Character String 4-27
draw at current position 4-27
draw at current position, with controls 4-30
draw at specified position 4-28
draw string at specified position, with controls 4-32
Character String At 4-28
Character String at Current Position 27-9
Character String at Given Position 27-9
Character String Extended at Current Position 27-9
Character String Extended at Given Position 27-9
Character String Move at Current Position 27-10
Character String Move at Given Position 27-10
Character String Position 4-30
Character String Position At 4-32
CHARBUNDLE 2-3
CHDIRN_* values 4-142, 4-256
checkbox 13-1
Chinese text 4-256
CHS_* values 4-30, 4-32, 4-146, 4-147
CLASSINFO 2-4
clipboard
query format information 9-182
query viewer window 9-184
set data 9-273
Clipboard messages 23-1
clipping 4-315, D-1
segment chains 4-72
set path 4-263
set region 4-265
clipping boundary 4-286
clipping region 9-95
Close Clipboard 9-25
Close Device Context 3-2
Close Figure 4-34, 27-11
Close Profile 6-4

Close Segment 4-35
closed figure 4-17
CLR_* values 4-51, 4-133, 4-151, 4-196, 4-244, 4-267
CMDSRC_* values 11-2, 12-16, 12-22, 12-44, 15-19
CM_* values 4-143, 4-252, 4-258
COBOL language v
code page
 query 9-185
 set 9-277
Code Page Change Hook 10-4
Code pages 28-1
 ASCII 28-7
 EBCDIC 28-12
 Font support 28-5
 options for VIO and PM applications 28-3
 support for multiple 28-5
CodePageChangeHook 10-4
color palette 9-221
color table D-1
 create 4-50
 realize 4-217
 unrealize 4-338
color table default values 4-51
colors
 on monochrome devices 4-52
 query 4-151
 query data 4-152
 query foreground mix mode 4-188
 query index 4-153
 query nearest 4-190
 query real 4-200
 query RGB 4-204
 query system 9-221
 realize logical table 4-217
 set 4-267
 set background 4-244
 set system values 9-295
Combine Region 4-37
combo box control data 19-1
combo box control window processing 19-1
Comment 4-38, 27-11
Compare Strings 9-26
COMPOSED 2-4
constant names 1-1
constants
 button filtering 9-112
contents and format of dialog template 26-18
control classes 11-1
control codes
 Shift In (SI) 28-18
 Shift Out (SO) 28-18
control data 26-21
control statements
 predefined 26-22
control window processing 11-1
conventions
 notation 1-1
Convert 4-39
coordinates
 dialog 26-18
coordinates for dialogs 26-18
Copy Accelerator Table 9-27
Copy Metafile 4-40
Copy Rectangle 9-28
Correlate Chain 4-41
Correlate From 4-44
Correlate Segment 4-46
cosmetic line width
 query 4-180
COUNT2 2-4
COUNT2B 2-4
COUNT2CH 2-4
COUNT4 2-4
COUNT4B 2-4
CPID 2-4
Create Accelerator Table 9-31
Create Atom Table 9-32
Create Bit Map 4-48
Create Cursor 9-33
Create Data Structure 9-35
Create Dialog 9-37
Create Frame Controls 9-38
Create Group 6-5, 9-39
Create Heap 9-41
Create Help Instance 9-43
Create Help Table 9-44
Create Logical Color Table 4-50
Create Logical Font 4-54
Create Menu 9-45
Create Message Queue 9-46
Create Pointer 9-47
Create Pointer Indirect 9-48
Create Presentation Space 4-56
Create Region 4-59
Create Standard Window 9-49
Create Switch Entry 9-51
Create Window 9-52
CREATEPARAMS 2-4
CREATESTRUCT 2-4
CREA_* values 4-112
CRGN_* values 4-37
CS_* values 9-116, 12-1
CTAB_* values 4-112
CTLDATA 2-5
current position
 move 4-100
 query 4-155
 set to specified point 4-270
cursor
 create 9-33
 destroy 9-67
 hide 9-310
 query information 9-187
 show 9-310
CURSORINFO 2-5
CURSOR_* values 9-33
CVR_* values 12-13
CVTC_* values 4-39

D

data
 bit map B-1
 get 4-87
 put 4-127
data area in a dialog template 26-21
data format
 image C-12
 outline C-12
data structures
 create 9-35
 decompose 9-188
 destroy 9-103
 modify contents 9-155

data structures *(continued)*
query 9-188
data types 2-1
graphics orders 27-1
metalanguage 2-1
DBCS strings 9-167
DBCS support
character-encoding schemes 28-18
ES_DBCS 14-1
DBM_* values 9-79
DB_* values 9-81
DCTL_* values 4-164, 4-279
DDEF_* values 4-112
DDEINIT 2-6
DDESTRUCT 2-6
DDE_* values 2-6, 24-1, 24-2, 24-3
Default AVio Window Procedure 9-59
Default Dialog Procedure 9-60
default dialog processing 12-50
default graphics character box
query 4-159
default message processing 12-1
default view matrix
query 4-158
Default Window Procedure 9-61
default window processing 11-1
DEFAULTICON keyword 26-10
Delete Atom 9-62
Delete Bit Map 4-60
Delete Element 4-61
Delete Element Range 4-62
Delete Elements Between Labels 4-63
Delete Library 9-63
Delete Metafile 4-64
Delete Procedure 9-64
Delete Segment 4-65
Delete Segments 4-66
Delete Set Identifier 4-67
Destroy Accelerator Table 9-65
Destroy Atom Table 9-66
Destroy Cursor 9-67
Destroy Data Structure 9-68
Destroy Group 6-7
Destroy Heap 9-69
Destroy Help Instance 9-70
Destroy Message Queue 9-71
Destroy Pointer 9-72
Destroy Presentation Space 4-68
Destroy Region 4-69
Destroy Window 9-73
detectability attribute for segments
modify (GpiSetSegmentAttrs) 4-322
DevCloseDC 3-2
DevEscape 3-3
DEVESC_* values 3-3, 3-4
device characteristics
query 3-14
device context
clear output display 4-81
close 3-2
create 3-10
open 3-10
open for a window 9-163
screen 9-85
DEVOPENDATA 2-6
DevOpenDC 3-10
DEVOPENSTRUC 2-6
DevPostDeviceModes 3-12
DevQueryCaps 3-14
DevQueryDeviceNames 3-19
DevQueryHardcopyCaps 3-21
DEV_* values 3-2, 3-11
DFORM_* values 4-87, 4-127
dialog
create 9-37
default procedure 9-60
dismiss 9-75
enumerate item 9-93
load 9-136
process modal 9-168
query item short 9-192
send message to item 9-265
set item short 9-278
dialog coordinates 26-18
dialog item
query text 9-193
query text length 9-194
set text 9-279
dialog points
map 9-150
Dialog Procedure 10-2
dialog processing
default 12-50
language support 12-56
dialog template 26-18
data-area information 26-21
format and contents 26-18
header information 26-19
item information 26-20
dialog window
destroy modal 9-75
hide modeless 9-75
DialogProc 10-2
dialogs
define procedure 10-2
directives 26-4
Dismiss Dialog 9-75
Dispatch Message 9-76
dithered colors 4-190
dithering 4-190, 9-295
DLGC_* values 12-52
DLGTEMPLATE 2-8
DLGTEMPLATE statement 26-15
DLGTITEM 2-8
DM_* values 4-165, 4-281
double-byte character sets 28-18
Down cursor key 9-324
DPDM_* values 3-13
DP_* values 9-83
Draw Bit Map 9-79
Draw Border 9-81
Draw Chain 4-70
Draw Dynamics 4-71
Draw From 4-72
Draw Pointer 9-83
Draw Segment 4-74
Draw Text 9-84
Draw Tracking Rectangle 9-323
drawing mode
draw 4-35, 4-76, 4-279, 4-282, 4-333
draw-and-retain 4-76, 4-167, 4-279, 4-282, 4-333
draw-and-retain mode 4-35
query 4-165
retain 4-76, 4-144, 4-167, 4-282, 4-333

drawing mode *(continued)*
set 4-281
drawing orders 27-1
DRIVDATA 2-9
DRO_* values 4-23, 4-85
DTYP_* values 9-232, 9-251, 9-301
DT_* values 9-84, 21-1
Dynamic Data Exchange Initiate 9-55
dynamic data exchange messages 24-1
Dynamic Data Exchange Post Message 9-56
Dynamic Data Exchange Respond 9-58

E

EBCDIC MIXED code pages 28-18
edit mode
query 4-166
set 4-283
EDI_* values 9-93
EGA 3-17
Element 4-76
end 4-78
query 4-167
elements
delete 4-61
delete between labels 4-63
delete between range 4-62
offset pointer 4-102
query pointer 4-168
query type 4-169
set pointer at label 4-285
Empty Clipboard 9-86
EM_CLEAR 14-4
EM_COPY 14-4
EM_CUT 14-5
EM_PASTE 14-5
EM_QUERYCHANGED 14-6
EM_QUERYFIRSTCHAR 14-7
EM_QUERYSEL 14-7
EM_SETFIRSTCHAR 14-8
EM_SETREADONLY 14-8
EM_SETSEL 14-9
EM_SETTEXTLIMIT 14-9
Enable Physical Input 9-87
Enable Window Update 9-89
End Area 4-77, 27-12
End Element 4-78, 27-12
End Image 27-12
End of Symbol Definition 27-13
End Paint 9-91
End Path 4-79, 27-13
End Prolog 27-13
End Window Enumeration 9-90
ENDFONT structure C-1
Enter key 9-324
entry field control data 14-2
entry field control window processing 14-1
ENTRYFDATA 14-2
Enumerate Clipboard Formats 9-92
Enumerate Dialog Item 9-93
EN_* values 14-3, 18-4
EQRGN_* values 4-80
Equal Rectangle 9-94
Equal Region 4-80
Erase 4-81
ERRINFO 2-9
Error Segment Data 4-82
error severities 1-2
error state
get last 9-110
error-information block 9-101
ERRORID 2-10
errors
explanations A-1
get information 9-108
severities of 1-2
Esc key 9-324
Escape 3-3, 27-14
ES_* values 14-1
Exclude Clip Rectangle 4-83
Exclude Update Region 9-95
Extended Escape 27-14

F

FATTRS 2-10
FATTR_FONTUSE_* values 2-11
FATTR_SEL_* values 2-10
FATTR_TYPE_* values 2-11
FCF_* values 9-262, 15-2
FC_* values 9-99
FFDESCS 2-11
FF_* indicators 9-246
FID_* values 15-1, 22-1
file format
file formats
bit maps B-2
icon file B-2
pointer B-2
Fill Path 4-84, 27-15
Fill Rectangle 9-96
Fillet at Current Position 27-15
Fillet at Given Position 27-15
Find Atom 9-97
FIXED 2-11
FI_* values 15-16
Flash Window 9-98
flashing
start 9-98
stop 9-98
flipping bits 9-127
FM_* values 4-188, 4-304
FOCAMETRICS structure C-2
focus
change window 9-99
query 9-195
set window 9-281
font character definitions C-8
font definition header C-8
font directory C-13
font metrics C-2
font-file format C-1
FONTDEFINITIONHEADER structure C-8
FONTMETRICS 2-11
fonts
create logical definition 4-54
definition of terms C-14
Japanese 28-18
load 4-95
outline 4-252, 4-254, 4-261
query 4-172
query metrics 4-171
query number of local identifiers 4-191

fonts *(continued)*
query set identifiers 4-209
query width table 4-216
raster 4-252, 4-254, 4-261, 4-310
unload 4-337
fonts supplied with OS/2 Version 1.2 C-12
FONTSIGNATURE structure C-1
FONT_* values 4-54
format
font-file C-1
format and contents of dialog template 26-18
FORTRAN language v
FPATH_* values 4-84
frame control data 15-3
frame control window processing 15-1
FRAMECDATA 15-3
Free Error Information 9-101
Free Memory on Heap 9-102
Free Message 9-103
FS_* values 15-3
Full Arc 4-85
create 4-85
Full Arc at Current Position 27-16
Full Arc at Given Position 27-16
function descriptions
conventions used 1-1
functions
supplied by applications 10-1

G

GARC 27-3
GBAR 27-3
GBBLT 27-6
GBEL 27-4
GBEZ 27-6
GBIMG 27-4
GBIT1 27-1
GBIT16 27-1
GBIT2 27-1
GBIT32 27-1
GBIT4 27-1
GBIT5 27-1
GBIT6 27-1
GBIT7 27-1
GBIT8 27-1
GBOX 27-7
GBPTH 27-5
GCALLS 27-8
GCARC 27-3
GCBEZ 27-6
GCBIMG 27-4
GCBOX 27-7
GCCHST 27-9
GCCHSTE 27-9
GCCHSTM 27-10
GCFARC 27-16
GCFLT 27-15
GCHAR 27-1
GCHST 27-9
GCHSTE 27-9
GCHSTM 27-10
GCLFIG 27-11
GCLINE 27-17
GCMRK 27-18
GCOMT 27-11
GCPARC 27-19
GCRLINE 27-20
GCSFLT 27-46
GDELPOINT 27-2
GEAR 27-12
GEEL 27-12
GEESCP 27-14
GEIMG 27-12
general window styles 12-1
geometric line width 4-181
GEPROL 27-13
GEPTH 27-13
GESCP 27-14
GESD 27-13
Get Clipped Presentation Space 9-105
Get Current Time 9-106
Get Data 4-87
Get Dialog Message 9-107
Get Error Information 9-108
Get Key State 9-109
Get Last Error 9-110
Get Message 9-112
Get Minimum Position 9-111
Get Multiple Windows From Identities 9-159
Get Next Window 9-114
Get Physical Key State 9-115
Get Presentation Space 9-116
Get Screen Presentation Space 9-117
Get System Bit Map 9-118
GFARC 27-16
GFIXED 27-2
GFLT 27-15
GFPTH 27-15
GHBITMAP 27-2
GIMD 27-16
GINDATT 27-2
GINDEX3 27-2
GLBL 27-17
GLENGTH1 27-2
GLENGTH2 27-2
GLINE 27-17
GLONG 27-2
Glyph list 28-5
GMPTH 27-18
GMRK 27-18
GNOP1 27-19
GOPTH 27-19
GPARC 27-19
GpiAssociate 4-10
GpiBeginArea 4-11
GpiBeginElement 4-14
GpiBeginPath 4-16
GpiBitBlt 4-19
GpiBox 4-23
GpiCallSegmentMatrix 4-25
GpiCharString 4-27
GpiCharStringAt 4-28
GpiCharStringPos 4-30
GpiCharStringPosAt 4-32
GpiCloseFigure 4-34
GpiCloseSegment 4-35
GpiCombineRegion 4-37
GpiComment 4-38
GpiConvert 4-39
GpiCopyMetaFile 4-40
GpiCorrelateChain 4-41
GpiCorrelateFrom 4-44
GpiCorrelateSegment 4-46

GpiCreateBitmap 4-48
GpiCreateLogColorTable 4-50
GpiCreateLogFont 4-54
GpiCreatePS 4-56
GpiCreateRegion 4-59
GpiDeleteBitmap 4-60
GpiDeleteElement 4-61
GpiDeleteElementRange 4-62
GpiDeleteElementsBetweenLabels 4-63
GpiDeleteMetaFile 4-64
GpiDeleteSegment 4-65
GpiDeleteSegments 4-66
GpiDeleteSetId 4-67
GpiDestroyPS 4-68
GpiDestroyRegion 4-69
GpiDrawChain 4-70
GpiDrawDynamics 4-71
GpiDrawFrom 4-72
GpiDrawSegment 4-74
GpiElement 4-76
GpiEndArea 4-77
GpiEndElement 4-78
GpiEndPath 4-79
GpiEqualRegion 4-80
GpiErase 4-81
GpiErrorSegmentData 4-82
GpiExcludeClipRectangle 4-83
GPIE_* values 4-82
GpiFillPath 4-84
GpiFullArc 4-85
GPIF_* values 4-318
GpiGetData 4-87
GpiImage 4-89
GpiIntersectClipRectangle 4-91
GpiLabel 4-92
GpiLine 4-93
GpiLoadBitmap 4-94
GpiLoadFonts 4-95
GpiLoadMetaFile 4-96
GpiMarker 4-97
GpiModifyPath 4-98
GpiMove 4-100
GpiOffsetClipRegion 4-101
GpiOffsetElementPointer 4-102
GpiOffsetRegion 4-103
GpiOpenSegment 4-104
GpiOutlinePath 4-106
GpiPaintRegion 4-107
GpiPartialArc 4-108
GpiPlayMetaFile 4-110
GpiPointArc 4-115
GpiPolyFillet 4-116
GpiPolyFilletSharp 4-118
GpiPolyLine 4-120
GpiPolyMarker 4-121
GpiPolySpline 4-122
GpiPop 4-124
GpiPtInRegion 4-125
GpiPtVisible 4-126
GpiPutData 4-127
GpiQueryArcParams 4-129
GpiQueryAttrMode 4-130
GpiQueryAttrs 4-131
GpiQueryBackColor 4-133
GpiQueryBackMix 4-134
GpiQueryBitmapBits 4-135
GpiQueryBitmapDimension 4-136
GpiQueryBitmapHandle 4-137
GpiQueryBitmapParameters 4-138
GpiQueryBoundaryData 4-139
GpiQueryCharAngle 4-140
GpiQueryCharBox 4-141
GpiQueryCharDirection 4-142
GpiQueryCharMode 4-143
GpiQueryCharSet 4-144
GpiQueryCharShear 4-145
GpiQueryCharStringPos 4-146
GpiQueryCharStringPosAt 4-147
GpiQueryClipBox 4-149
GpiQueryClipRegion 4-150
GpiQueryColor 4-151
GpiQueryColorData 4-152
GpiQueryColorIndex 4-153
GpiQueryCp 4-154
GpiQueryCurrentPosition 4-155
GpiQueryDefArcParams 4-156
GpiQueryDefAttrs 4-157
GpiQueryDefaultViewMatrix 4-158
GpiQueryDefCharBox 4-159
GpiQueryDefTag 4-160
GpiQueryDefViewingLimits 4-161
GpiQueryDevice 4-162
GpiQueryDeviceBitmapFormats 4-163
GpiQueryDrawControl 4-164
GpiQueryDrawingMode 4-165
GpiQueryEditMode 4-166
GpiQueryElement 4-167
GpiQueryElementPointer 4-168
GpiQueryElementType 4-169
GpiQueryFontFileDescriptions 4-170
GpiQueryFontMetrics 4-171
GpiQueryFonts 4-172
GpiQueryGraphicsField 4-174
GpiQueryInitialSegmentAttrs 4-175
GpiQueryKerningPairs 4-176
GpiQueryLineEnd 4-177
GpiQueryLineJoin 4-178
GpiQueryLineType 4-179
GpiQueryLineWidth 4-180
GpiQueryLineWidthGeom 4-181
GpiQueryLogColorTable 4-182
GpiQueryMarker 4-183
GpiQueryMarkerBox 4-184
GpiQueryMarkerSet 4-185
GpiQueryMetaFileBits 4-186
GpiQueryMetaFileLength 4-187
GpiQueryMix 4-188
GpiQueryModelTransformMatrix 4-189
GpiQueryNearestColor 4-190
GpiQueryNumberSetIds 4-191
GpiQueryPageViewport 4-192
GpiQueryPattern 4-193
GpiQueryPatternRefPoint 4-194
GpiQueryPatternSet 4-195
GpiQueryPel 4-196
GpiQueryPickAperturePosition 4-197
GpiQueryPickApertureSize 4-198
GpiQueryPS 4-199
GpiQueryRealColors 4-200
GpiQueryRegionBox 4-202
GpiQueryRegionRects 4-203
GpiQueryRGBColor 4-204
GpiQuerySegmentAttrs 4-205
GpiQuerySegmentNames 4-206

GpiQuerySegmentPriority 4-207
GpiQuerySegmentTransformMatrix 4-208
GpiQuerySetIds 4-209
GpiQueryStopDraw 4-210
GpiQueryTag 4-211
GpiQueryTextBox 4-212
GpiQueryViewingLimits 4-214
GpiQueryViewingTransformMatrix 4-215
GpiQueryWidthTable 4-216
GpiRealizeColorTable 4-217
GpiRectInRegion 4-218
GpiRectVisible 4-219
GpiRemoveDynamics 4-220
GpiResetBoundaryData 4-222
GpiResetPS 4-223
GpiRestorePS 4-225
GpiRotate 4-226
GpiSaveMetaFile 4-228
GpiSavePS 4-229
GpiScale 4-231
GpiSetArcParams 4-233
GpiSetAttrMode 4-235
GpiSetAttrs 4-237
GpiSetBackColor 4-244
GpiSetBackMix 4-246
GpiSetBitmap 4-248
GpiSetBitmapBits 4-249
GpiSetBitmapDimension 4-250
GpiSetBitmapId 4-251
GpiSetCharAngle 4-252
GpiSetCharBox 4-254
GpiSetCharDirection 4-256
GpiSetCharMode 4-258
GpiSetCharSet 4-260
GpiSetCharShear 4-261
GpiSetClipPath 4-263
GpiSetClipRegion 4-265
GpiSetColor 4-267
GpiSetCp 4-269
GpiSetCurrentPosition 4-270
GpiSetDefArcParams 4-271
GpiSetDefAttrs 4-272
GpiSetDefaultViewMatrix 4-275
GpiSetDefTag 4-277
GpiSetDefViewingLimits 4-278
GpiSetDrawControl 4-279
GpiSetDrawingMode 4-281
GpiSetEditMode 4-283
GpiSetElementPointer 4-284
GpiSetElementPointerAtLabel 4-285
GpiSetGraphicsField 4-286
GpiSetInitialSegmentAttrs 4-287
GpiSetLineEnd 4-289
GpiSetLineJoin 4-291
GpiSetLineType 4-293
GpiSetLineWidth 4-295
GpiSetLineWidthGeom 4-296
GpiSetMarker 4-298
GpiSetMarkerBox 4-300
GpiSetMarkerSet 4-302
GpiSetMetaFileBits 4-303
GpiSetMix 4-304
GpiSetModelTransformMatrix 4-306
GpiSetPageViewport 4-308
GpiSetPattern 4-309
GpiSetPatternRefPoint 4-311
GpiSetPatternSet 4-313
GpiSetPel 4-315
GpiSetPickAperturePosition 4-316
GpiSetPickApertureSize 4-317
GpiSetPS 4-318
GpiSetRegion 4-320
GpiSetSegmentAttrs 4-321
GpiSetSegmentPriority 4-323
GpiSetSegmentTransformMatrix 4-325
GpiSetStopDraw 4-327
GpiSetTag 4-328
GpiSetViewingLimits 4-329
GpiSetViewingTransformMatrix 4-331
GpiStrokePath 4-333
GpiTranslate 4-335
GpiUnloadFonts 4-337
GpiUnrealizeColorTable 4-338
GpiWCBitBlt 4-339
GPI_* values 4-113
GPOINT 27-2
GPOINTB 27-2
GPOP 27-20
GPSAP 27-21
GPSBCOL 27-22
GPSBICOL 27-22
GPSBMX 27-23
GPSCA 27-24
GPSCC 27-25
GPSCD 27-26
GPSCH 27-27
GPSCOL 27-29
GPSCP 27-29
GPSCR 27-26
GPSCS 27-27
GPSECOL 27-30
GPSFLW 27-30
GPSIA 27-32
GPSICOL 27-31
GPSLE 27-33
GPSLJ 27-33
GPSLT 27-34
GPSLW 27-35
GPSMC 27-35
GPSMP 27-36
GPSMS 27-36
GPSMT 27-37
GPSMX 27-38
GPSPIK 27-42
GPSPRP 27-40
GPSPS 27-40
GPSPT 27-41
GPSSLW 27-44
GPSTM 27-39
GPSVW 27-45
GRADIENT 2-17
GRADIENTL 2-17
graphics
 orders 27-1
 query field 4-174
 set field 4-286
graphics orders
 data types 27-1
GREAL 27-2
GRES_* values 4-223
GRLINE 27-20
GROF 27-2
GROFUFS 27-2
GROSOL 27-2

GROUFS 27-2
GSAP 27-21
GSBCOL 27-22
GSBICOL 27-22
GSBMX 27-23
GSCA 27-24
GSCC 27-25
GSCD 27-26
GSCH 27-27
GSCOL 27-29
GSCP 27-29
GSCPTH 27-28
GSCR 27-26
GSCS 27-27
GSECOL 27-30
GSFLT 27-46
GSFLW 27-30
GSGCH 27-43
GSHORT 27-2
GSIA 27-32
GSICOL 27-31
GSLE 27-33
GSLJ 27-33
GSLT 27-34
GSLW 27-35
GSMC 27-35
GSMP 27-36
GSMS 27-36
GSMT 27-37
GSMX 27-38
GSPIK 27-42
GSPRP 27-40
GSPS 27-40
GSPT 27-41
GSSB 27-42
GSSLW 27-44
GSTM 27-39
GSTR 27-2
GSTV 27-44
GSVW 27-45
GUCHAR 27-2
GUFIXEDS 27-2
GULONG 27-2
GUSHORT370 27-2

H

HAB 2-17
HACCEL 2-17
HANDLE 2-17
HAPP 2-17
HATOMTBL 2-17
HBITMAP 2-17
HCAPS_* values 2-17
HCINFO 2-17
HDC 2-18
header 26-19
Help Hook 10-5
help manager messages 25-1
HelpHook 10-5
HELPINIT 2-18
HELPSUBTABLE 2-19
HELPSUBTABLEITEM 2-19
HELPTABLE 2-19
HENUM 2-20
HFM_* values 10-5
HHEAP 2-20
HIGHER_* values 4-207, 4-323
highlight attribute for segments
 modify (GpiSetSegmentAttrs) 4-322
HINI 2-20
HK_* values 9-282
HLIB 2-20
HMF 2-20
HMODULE 2-20
HMQ 2-20
HMQ_* values 9-259
HM_ACTION_BAR_COMMAND 25-1
HM_CREATE_HELP_TABLE 25-1
HM_DISMISS_WINDOW 25-2
HM_DISPLAY_HELP 25-2
HM_ERROR 25-3
HM_EXT_HELP 25-4
HM_EXT_HELP_UNDEFINED 25-5
HM_HELPSUBITEM_NOT_FOUND 25-6
HM_HELP_CONTENTS 25-5
HM_HELP_INDEX 25-6
HM_INFORM 25-7
HM_KEYS_HELP 25-8
HM_LOAD_HELP_TABLE 25-8
HM_QUERY_KEYS_HELP 25-9
HM_REPLACE_HELP_FOR_HELP 25-9
HM_SET_ACTIVE_WINDOW 25-10
HM_SET_HELP_LIBRARY_NAME 25-11
HM_SET_HELP_WINDOW_TITLE 25-11
HM_SET_SHOW_PANEL_ID 25-12
HM_TUTORIAL 25-12
HM_* values 9-41
hook
 change code page 10-4
 help requests 10-5
 input 10-8
 message filter 10-15
 release 9-259
 send message 10-18
 set 9-282
hooks 10-1
HPOINTER 2-20
HPROGARRAY 2-20
HPROGRAM 2-20
HPS 2-20
HRGN 2-20
HRGN_* values 4-265
HSEM 2-20
HSPL 2-20
HSWITCH 2-20
HT_* values 12-23
HVPS 2-20
HWND 2-20
HWND_* values 9-13, 9-37, 9-38, 9-45, 9-77, 9-136, 9-140, 9-151, 9-221, 9-303

I

IBB_* values 4-238, 4-273
icon
 destroy 9-72
icon file format B-2
IDENTITY 2-20
IDENTITY4 2-20
Image 4-89
 draw 4-89
image attribute values 4-238, 4-273

Image Data 27-16
image data format C-12
IMAGEBUNDLE 2-20
Implicit Pointer 1-1
In Send Message 9-121
INDEX2 2-20
Inflate Rectangle 9-119
information tables
 bit map B-1
initialization file E-1
Initialize 9-120
initialize Presentation Interface 9-120
Input Hook 10-8
InputHook 10-8
integer values
 create 9-301
 decompose 9-232
 query 9-232
 set 9-301
interchange file format D-1
Intersect Clip Rectangle 4-91
Intersect Rectangle 9-124
Invalidate Rectangle 9-125
Invalidate Region 9-126
Invert Rectangle 9-127
IPT 2-20
Is Child 9-128
Is Rectangle Empty 9-129
Is Thread Active 9-130
Is Window 9-131
items in a dialog template 26-20

J

Japanese fonts 28-18
Journal Playback Hook 10-9
Journal Record Hook 10-10
JournalPlaybackHook 10-9
JournalRecordHook 10-10
JRN_* values 12-24

K

kanji 28-18
KC_* values 12-14
kerning 2-16, 2-20
 device support 3-17
 enable 2-11
 number of pairs 2-16
 query pairs 4-176
kerning pair table C-13
KERNINGPAIRS 2-20
Keyboard control codes 12-14
keyboard resources 26-17
keyboard statements
 keyboard 26-17

L

Label 4-92, 27-17
 generate element for 4-92
language bindings v
language support dialog processing 12-56
languages
 IBM COBOL/2 v
 IBM C/2 v
 IBM FORTRAN/2 v
languages *(continued)*
 IBM Macro Assembler/2 v
LBB_* values 4-237, 4-272
LCIDT_* values 4-209
LCID_* values 4-144, 4-185, 4-195, 4-260, 4-302, 4-313
LCOLF_* values 4-50, 4-152, 9-295
LCOLOPT_* values 4-153, 4-182, 4-190, 4-200, 4-204
LCOL_* values 4-50, 9-295
LC_* values 4-111
Left cursor key 9-324
LENGTH1 2-20
LENGTH2 2-21
LENGTH4 2-21
LHANDLE 2-21
Line 4-93
 draw 4-93
 query cosmetic width 4-180
 query end 4-177
 query geometric width 4-181
 query join 4-178
 query type 4-179
 query width 4-180
 set cosmetic width 4-295
 set end 4-289
 set geometric width 4-296
 set join 4-291
 set type 4-293
 set width 4-295
Line at Current Position 27-17
Line at Given Position 27-17
line attribute values 4-237, 4-272
LINEBUNDLE 2-21
LINEEND_* values 4-177, 4-289
LINEJOIN_* values 4-178, 4-291
LINETYPE_* values 4-179, 4-293
LINEWIDTHGEOM_* values 4-181
LINEWIDTH_* values 4-180, 4-295
list box control data 16-1
list box control styles 16-1
list box control window processing 16-1
LIT_* values 16-6
LM_DELETEALL 16-5
LM_DELETEITEM 16-5
LM_INSERTITEM 16-6
LM_QUERYCURSOR 16-7
LM_QUERYITEMCOUNT 16-7
LM_QUERYITEMHANDLE 16-8
LM_QUERYITEMTEXT 16-8
LM_QUERYITEMTEXTLENGTH 16-9
LM_QUERYSELECTION 16-9
LM_QUERYTOPINDEX 16-10
LM_SEARCHSTRING 16-11
LM_SELECTITEM 16-12
LM_SETCURSOR 16-12
LM_SETITEMHANDLE 16-13
LM_SETITEMHEIGHT 16-14
LM_SETITEMTEXT 16-14
LM_SETSELECTION 16-15
LM_SETTOPINDEX 16-15
LN_* values 16-2
Load Accelerator Table 9-135
Load and Process Modal Dialog 9-77
Load Bit Map 4-94
Load Dialog 9-136
Load Fonts 4-95
Load Help Table 9-138

Load Library 9-139
Load Menu 9-140
Load Metafile 4-96
Load Pointer 9-141
Load Procedure 9-142
Load String 9-143
load type options 4-110
Loader Hook 10-11
LoaderHook 10-11
LOADOPTION 26-2
local identifier options 4-111
Lock heap 9-144
Lock Visible Regions 9-145
Lock Window 9-146
Lock Window Update 9-147
logical color table
 create 4-50
 realize 4-217
 unrealize 4-338
logical font
 delete 4-67
LONG 2-21
LOWER_* values 4-207, 4-323
LSS_* values 16-11
LS_* values 16-1
LT_* values 4-110

M

Macro Assembler language v
Make Points 9-148
Make Rectangle 9-149
Map Dialog Points 9-150
Map Window Points 9-151
MARGSTRUCT 2-21
Marker 4-97
 draw a series of 4-121
 draw with center at specified position 4-97
 query 4-183
 query box 4-184
 query set 4-185
 query symbol 4-183
 set 4-298
 set box 4-300
 set set 4-302
Marker at Current Position 27-18
Marker at Given Position 27-18
marker attribute values 4-238, 4-272
MARKERBUNDLE 2-21
MARKSYM_* values 4-183, 4-298
MASM v
MATRIX 2-22
MATRIXLF 2-22
MBB_* values 4-272
MBID_* values 9-153
MB_* values 9-152, 9-153
MEMOPTION 26-2
memory
 release 9-101
menu control styles 17-1
menu control window processing 17-1
menu item attributes 17-2
MENU item definition statements 26-12
menu item styles 17-2
MENU statement 26-11
MENUITEM 2-23
menus
 create 9-45
 create window 9-45
 load 9-140
 pull-down 26-13
 templates 26-14
message
 broadcast 9-19
 dispatch 9-76
Message Box 9-152
Message Control Hook 10-13
Message Filter Hook 10-15
Message or Multiple Semaphore Wait 9-157
Message or Semaphore Wait 9-158
message processing
 introduction 11-1
 notation conventions 11-3
 types 11-1
message types 11-1
messages
 create queue 9-46
 destroy queue 9-71
 get one 9-112
 peek 9-164
 post 9-165
 post queue 9-166
 send 9-266
 wait for 9-333
Metafile data format D-2
metafile restrictions D-1
metafiles
 create new 4-40
 delete 4-64
 general rules D-1
 load 4-96
 play 4-110
 query bits 4-186
 query length 4-187
 SAA-conforming 4-271, 4-274, 4-277, 4-278
 save 4-228
metalanguage 1-1
metalanguage data types 2-1
MIA_* values 17-2
micro-presentation space 4-229, 4-279
MIS_* values 17-2, 26-14
MIT_* values 17-10, 17-12, 17-17
mix
 query 4-188
 set 4-304
 set background 4-246
 set foreground 4-304
MIXED strings 28-18
MLECTLDATA 18-2
MLE_SEARCHDATA 2-23
MLM_CHARFROMLINE 18-9
MLM_CLEAR 18-8
MLM_COPY 18-8
MLM_CUT 18-9
MLM_DELETE 18-10
MLM_DISABLEREFRESH 18-10
MLM_ENABLEREFRESH 18-11
MLM_EXPORT 18-12
MLM_FORMAT 18-12
MLM_IMPORT 18-13
MLM_INSERT 18-14
MLM_LINEFROMCHAR 18-14

MLM_PASTE 18-15
MLM_QUERYBACKCOLOR 18-15
MLM_QUERYCHANGED 18-16
MLM_QUERYFIRSTCHAR 18-16
MLM_QUERYFONT 18-17
MLM_QUERYFORMATLINELENGTH 18-17
MLM_QUERYFORMATRECT 18-18
MLM_QUERYFORMATTEXTLENGTH 18-18
MLM_QUERYIMPORTEXPORT 18-19
MLM_QUERYLINECOUNT 18-19
MLM_QUERYLINELENGTH 18-20
MLM_QUERYREADONLY 18-20
MLM_QUERYSEL 18-21
MLM_QUERYSELTEXT 18-22
MLM_QUERYTABSTOP 18-22
MLM_QUERYTEXTCOLOR 18-23
MLM_QUERYTEXTLENGTH 18-23
MLM_QUERYTEXTLIMIT 18-24
MLM_QUERYUNDO 18-24
MLM_QUERYWRAP 18-25
MLM_RESETUNDO 18-25
MLM_SEARCH 18-26
MLM_SETBACKCOLOR 18-27
MLM_SETCHANGED 18-28
MLM_SETFIRSTCHAR 18-28
MLM_SETFONT 18-29
MLM_SETFORMATRECT 18-29
MLM_SETIMPORTEXPORT 18-30
MLM_SETREADONLY 18-31
MLM_SETSEL 18-31
MLM_SETTABSTOP 18-32
MLM_SETTEXTCOLOR 18-32
MLM_SETTEXTLIMIT 18-33
MLM_SETWRAP 18-33
MLM_UNDO 18-34
MLS_* values 18-2
MM_DELETEITEM 17-8
MM_DISMISSMENU 17-8
MM_ENDMENUMODE 17-9
MM_INSERTITEM 17-9
MM_ISITEMVALID 17-10
MM_ITEMIDFROMPOSITION 17-11
MM_ITEMPOSITIONFROMID 17-11
MM_QUERYITEM 17-12
MM_QUERYITEMATTR 17-13
MM_QUERYITEMCOUNT 17-13
MM_QUERYITEMTEXT 17-14
MM_QUERYITEMTEXTLENGTH 17-15
MM_QUERYSELITEMID 17-15
MM_REMOVEITEM 17-16
MM_SELECTITEM 17-17
MM_SETITEM 17-18
MM_SETITEMATTR 17-18
MM_SETITEMHANDLE 17-19
MM_SETITEMTEXT 17-20
MM_STARTMENUMODE 17-20
modal dialog
 load and process 9-77
Modify Data Structure 9-155
Modify Path 4-98, 27-18
monochrome devices 4-180
Move 4-100
Move to Next Character 9-160
Move to Previous Character 9-167
MPARAM 2-23
MPATH_* values 4-98
MQINFO 2-24
MRESULT 2-23
MsgCtlHook 10-13
MsgFilterHook 10-15
MSGF_* values 10-15
MS_* values 12-4, 17-1
MT 2-24
MTI 2-24
multi-line entry field control data 18-2
multi-line entry field control window processing 18-1
multiple-line statements 26-7
 ACCELTABLE 26-8
 ASSOCTABLE 26-10
 DLGTEMPLATE 26-15
 MENU 26-11
 STRINGTABLE 26-7
 WINDOWTEMPLATE 26-15

N

No-Operation 27-19
nonstore attribute for segments
 modify (GpiSetSegmentAttrs) 4-322
notation conventions 1-1
 messages 11-3
NO_* values 8-3
NULL 1-1

O

Offset Clip Region 4-101
Offset Element Pointer 4-102
Offset Rectangle 9-161
Offset Region 4-103
OFFSET2B 2-24
Open Clipboard 9-162
Open Device Context 3-10
open figure 4-17
Open Profile 6-8
Open Segment 4-104
Open Window Device Context 9-163
outline data format C-12
outline fonts 4-252, 4-254, 4-259, 4-261
Outline Path 4-106, 27-19
OVERFLOW 2-24
owner-notification messages 11-2
OWNERITEM 2-25

P

page viewport
 query 4-192
 set 4-308
paint
 begin 9-18
 end 9-91
Paint Region 4-107
PARAM 2-25
parent/child/owner relationship 26-22
Partial Arc 4-108
Partial Arc at Current Position 27-19
Partial Arc at Given Position 27-19
path
 begin 4-16
 draw interior 4-84
 draw outline 4-106

path *(continued)*
end 4-79
fill 4-84
modify 4-98
PATSYM_* values 4-193, 4-309
pattern
query 4-193
pattern attribute (area) values 4-238, 4-272
patterns
query reference point 4-194
query set 4-195
set 4-309
set reference point 4-311
set set 4-313
PC VKEY F-1
Peek Message 9-164
pel
query 4-196
set 4-315
PFOCAMETRICS type C-2
PIBSTRUCT 2-27
PicIchg 5-2
pick aperture
query size 4-198
set size 4-317
PICKAP_* values 4-317
PICKSEL_* values 4-41, 4-44, 4-46
PicPrint 5-4
Picture Interchange Convert 5-2
Picture Print 5-4
PID 2-27
pie
segment 4-109
PIX 2-27
Play Metafile 4-110
PL_ALTERED 12-3
PMF_* values 4-110
PM_Q_* values 2-7
PM_* application names E-1
PM_* values 9-164, 10-8
POINT 2-27
Point Arc 4-115
Point In Rectangle 9-169
Point In Region 4-125
Point Visible 4-126
pointer
create 9-47
create indirect 9-48
destroy 9-72
draw 9-83
hide 9-311
implicit 1-1
load 9-141
query handle 9-200
query information 9-201
query position 9-202
set 9-289
set element 4-284
set position 9-290
show 9-311
pointer file format B-2
POINTERINFO 2-27
pointing device
capture messages 9-271
POINTL 2-27
POINTS 2-28
check whether visible 4-126
POINTS *(continued)*
check whether within region 4-125
Polyfillet 4-116
draw 4-116
sharp 4-118
Polyfillet Sharp 4-118
Polyline 4-120
draw 4-120
Polymarker 4-121
Polyspline 4-122
Pop 4-124, 27-20
Post Device Modes 3-12
Post Message 9-165
Post Queue Message 9-166
predefined control statements 26-22
PRESDATA 2-28
Presentation Interface
initialize 9-120
Presentation Manager
query environment 9-234
query revision level 9-234
query version 9-234
presentation parameters 26-21
presentation space
cache 9-18
cached 15-9
create 4-56
destroy 4-68
get a cache 9-116
micro 4-58, 9-80, 9-82, 9-85, 9-116
normal 9-80, 9-82, 9-85
options 4-56, 4-318
query 4-199
release cache 9-260
reset 4-223
restore 4-225
save 4-229
presentation space options 4-56, 4-318
PRESPARAMS 2-28
PrfAddProgram 6-2
PrfChangeProgram 6-3
PrfCloseProfile 6-4
PrfCreateGroup 6-5
PrfDestroyGroup 6-7
PrfOpenProfile 6-8
PRFPROFILE 2-28
PrfQueryDefinition 6-9
PrfQueryProfile 6-11
PrfQueryProfileData 6-12
PrfQueryProfileInt 6-14
PrfQueryProfileSize 6-15
PrfQueryProfileString 6-17
PrfQueryProgramCategory 6-19
PrfQueryProgramHandle 6-20
PrfQueryProgramTitles 6-21
PrfRemoveProgram 6-23
PrfReset 6-24
PrfWriteProfileData 6-25
PrfWriteProfileString 6-26
PRGN_* values 4-125
primitives
set attributes for 4-237
PRIM_* values 4-131, 4-157, 4-237, 4-272
PROC 2-28
procedures 10-1
dialog 10-2
window 10-3

Process Modal Dialog 9-168
profile
 query string 6-17, 9-210
PROGCATEGORY 2-28
PROGDETAILS 2-28
PROGRAMENTRY 2-29
PROGTITLE 2-29
PROGTYPE 2-29
PROG_* values 2-29
prompted entry field control window processing 19-1
PROPERTY2 2-29
PROPERTY4 2-29
PSF_* values 9-105
PSZ 2-29
PS_* values 4-56, 4-199, 4-318
pull-down menus 26-13
Push and Set Arc Parameter 27-21
Push and Set Background Color 27-22
Push and Set Background Indexed Color 27-22
Push and Set Background Mix 27-23
Push and Set Character Angle 27-24
Push and Set Character Cell 27-25
Push and Set Character Direction 27-26
Push and Set Character Precision 27-26
Push and Set Character Set 27-27
Push and Set Character Shear 27-27
Push and Set Color 27-29
Push and Set Current Position 27-29
Push and Set Extended Color 27-30
Push and Set Fractional Line Width 27-30
Push and Set Indexed Color 27-31
Push and Set Individual Attribute 27-32
Push and Set Line End 27-33
Push and Set Line Join 27-33
Push and Set Line Type 27-34
Push and Set Line Width 27-35
Push and Set Marker Cell 27-35
Push and Set Marker Precision 27-36
Push and Set Marker Set 27-36
Push and Set Marker Symbol 27-37
Push and Set Mix 27-38
Push and Set Model Transform 27-39
Push and Set Pattern Reference Point 27-40
Push and Set Pattern Set 27-40
Push and Set Pattern Symbol 27-41
Push and Set Pick Identifier 27-42
Push and Set Stroke Line Width 27-44
Push and Set Viewing Window 27-45
Put Data 4-127
PU_* values 4-56, 4-318
PVIS_* values 4-126
PVOID 2-29

Q

QCD_LCT_* values 4-152
QFC_* values 15-14
QF_* values 4-172, 8-10
QLCT_* values 4-182
QMOPENDATA 2-29
QMSG 2-29, 11-1
QS_* values 9-215
Query Accelerator Table 9-170
Query Active Window 9-171
Query Anchor Block 9-172
Query Arc Parameters 4-129
Query Atom Length 9-173
Query Atom Name 9-174
Query Atom Usage 9-175
Query Attribute Mode 4-130
Query Attributes 4-131
Query Available Heap Space 9-16
Query Background Color 4-133
Query Background Mix 4-134
Query Bit-Map Bits 4-135
Query Bit-Map Dimension 4-136
Query Bit-Map Handle 4-137
Query Bit-Map Parameters 4-138
Query Bits 9-176
Query Bits Under Mask 9-177
Query Boundary Data 4-139
Query Capture 9-178
Query Character Angle 4-140
Query Character Box 4-141
Query Character Direction 4-142
Query Character Mode 4-143
Query Character Set 4-144
Query Character Shear 4-145
Query Character String Positions 4-146
Query Character String Positions At 4-147
Query Class Information 9-179
Query Class Name 9-180
Query Clip Box 4-149
Query Clip Region 4-150
Query Clipboard Data 9-181
Query Clipboard Format Information 9-182
Query Clipboard Owner 9-183
Query Clipboard Viewer 9-184
Query Code Page 4-154, 9-185
Query Code Page List 9-186
Query Color 4-151
Query Color Data 4-152
Query Color Index 4-153
Query Current Position 4-155
Query Cursor Information 9-187
Query Data Structure 9-188
Query Default Arc Parameters 4-156
Query Default Attributes 4-157
Query Default Graphics Character Box 4-159
Query Default Tag 4-160
Query Default View Matrix 4-158
Query Default Viewing Limits 4-161
Query Definition 6-9, 9-190
Query Desktop Window 9-191
Query Device 4-162
Query Device Bit-Map Formats 4-163
Query Device Capabilities 3-14
Query Device Names 3-19
Query Dialog Item Short 9-192
Query Dialog Item Text 9-193
Query Dialog Item Text Length 9-194
Query Draw Control 4-164
Query Drawing Mode 4-165
Query Edit Mode 4-166
Query Element 4-167
Query Element Pointer 4-168
Query Element Type 4-169
Query Focus 9-195
Query Font File Descriptions 4-170
Query Font Metrics 4-171
Query Font Width Table 4-216
Query Fonts 4-172
Query Graphics Field 4-174

Query Hardcopy Caps 3-21
Query Help Instance 9-196
Query Initial Segment Attributes 4-175
Query Kerning Pairs 4-176
Query Line End 4-177
Query Line Join 4-178
Query Line Type 4-179
Query Line Width 4-180
Query Line Width Geom 4-181
Query Logical Color Table 4-182
Query Marker 4-183
Query Marker Box 4-184
Query Marker Set 4-185
Query Message Position 9-197
Query Message Time 9-198
Query Metafile Bits 4-186
Query Metafile Length 4-187
Query Mix 4-188
Query Model Transform Matrix 4-189
Query Nearest Color 4-190
Query Number Set Identifiers 4-191
Query Object Window 9-199
Query Page Viewport 4-192
Query Pattern 4-193
Query Pattern Reference Point 4-194
Query Pattern Set 4-195
Query Pel 4-196
Query Pick Aperture Position 4-197
Query Pick Aperture Size 4-198
Query Pointer 9-200
Query Pointer Information 9-201
Query Pointer Position 9-202
Query Presentation Parameter 9-203
Query Presentation Space 4-199
Query Profile 6-11
Query Profile Data 6-12, 9-205
Query Profile Integer 6-14, 9-207
Query Profile Size 6-15, 9-208
Query Profile String 6-17, 9-210
Query Program Category 6-19
Query Program Handle 6-20
Query Program Titles 6-21, 9-212
Query Queue Information 9-214
Query Queue Status 9-215
Query Real Colors 4-200
Query Region Box 4-202
Query Region Rectangles 4-203
Query RGB Color 4-204
Query Segment Attributes 4-205
Query Segment Names 4-206
Query Segment Priority 4-207
Query Segment Transform Matrix 4-208
Query Session Title 9-217
Query Set Identifiers 4-209
Query Stop Draw 4-210
Query Switch Entry 9-218
Query Switch Handle 9-219
Query Switch List 9-220
Query System Atom Table 9-227
Query System Color 9-221
Query System Modal Window 9-222
Query System Pointer 9-223
Query System Value 9-224
Query Tag 4-211
Query Task Title 9-229
Query Task Window Size and Position 9-228
Query Text Box 4-212
Query Update Rectangle 9-230
Query Update Region 9-231
Query Value 9-232
Query Version 9-234
Query Viewing Limits 4-214
Query Viewing Transform Matrix 4-215
Query Window 9-235
Query Window Device Context 9-237
Query Window Enabled State 9-132
Query Window Handle From Device Context 9-334
Query Window Handle From Identifier 9-335
Query Window Lock Count 9-238
Query Window Long 9-245
Query Window Pointer 9-241
Query Window Position 9-239
Query Window Process 9-240
Query Window Rectangle 9-242
Query Window Short 9-246
Query Window Showing 9-133
Query Window Text 9-243
Query Window Text Length 9-244
Query Window Visibility 9-134
queue
 query information 9-214
 query status 9-215
QV_* values 9-234
QWL_* values 9-245
QWS_* values 9-246
QW_*values 9-235

R

radio button 13-1
raster fonts 4-252, 4-254, 4-259, 4-261
Realize Color Table 4-217
Reallocate Memory In Heap 9-248
RECT 2-30
rectangle
 calculate frame 9-20
 check whether visible 4-219
 check whether within region 4-218
 compare for equality 9-94
 convert to graphic 9-149
 copy 9-28
 draw border 9-81
 draw interior 9-81
 exclude from clipping region 4-83
 fill 9-96
 inflate 9-119
 intersect 9-124
 intersect clip 4-91
 invalidate 9-125
 invert 9-127
 query if point within 9-169
 query update 9-230
 set coordinates 9-292
 set empty 9-293
 subtract 9-318
 validate 9-331
Rectangle In Region 4-218
Rectangle Visible 4-219
RECTDIR_* values 2-31
RECTL 2-30
region
 query box 4-202
 query rectangles 4-203
regions

regions *(continued)*
check if identical 4-80
check whether point within 4-125
check whether rectangle within 4-218
combine 4-37
create 4-59
destroy 4-69
invalidate 9-126
move 4-103
offset 4-103
paint 4-107
set 4-320
validate 9-332
Register User Data Type 9-251
Register User Message 9-256
Register User Message Hook 10-16
Register Window Class 9-249
Register Window Destroy 9-258
RegisterUserMsg 10-16
Relative Line at Current Position 27-20
Relative Line at Given Position 27-20
Release Hook 9-259
Release Presentation Space 9-260
Remove Dynamics 4-220
Remove Presentation Parameter 9-261
Remove Program Definition 6-23
Remove Switch Entry 9-262
reserved messages 12-1
reserved program group handles 9-11
Reset Boundary Data 4-222
reset options 4-111
Reset Presentation Manager 6-24
Reset Presentation Space 4-223
RESID 2-30
resource
load string from 9-143
resource definitions 26-2
resource file specification 26-25
resource files 26-1
definitions 26-2
introduction 26-1
source file specification 26-25
syntax definitions 26-1
resource script file
specification 26-2
resource script file specification
keyboard resources 26-17
user-defined resources 26-3
resource statements
ACCELTABLE 26-8
ASSOCTABLE 26-10
dialog template 26-15
directives 26-4
DLGTEMPLATE 26-15
MENU item definition 26-12
MENU statement 26-11
multiple-line 26-7
single line 26-2
STRINGTABLE 26-7
user-defined 26-3
window template 26-15
WINDOWTEMPLATE 26-15
Restore Execution Environment 9-322
Restore Presentation Space 4-225
RES_* values 4-111
RGB 2-30, 4-53
RGB (red-green-blue) 4-152, 4-200, 4-267, 9-221
RGB (red-green-blue) *(continued)*
query color 4-204
RGNRECT 2-30
RGN_* values 4-83, 4-91, 4-202, 4-265, 9-231
Right cursor key 9-324
ROF 2-31
ROL 2-31
Roman text 4-256
ROP_* values 4-20, 4-339
Rotate Transform 4-226
RRGN_* values 4-218
RT_* values 26-25
RUM_* values 9-256
RVIS_* values 4-219

S

SAA-conforming metafiles 4-280
Save Execution Environment 9-23
Save Metafile 4-228
Save Presentation Space 4-229
SBCDATA 20-1
SBCS 28-18
SBMP_* values 9-118
SBM_QUERYHILITE 20-4
SBM_QUERYPOS 20-4
SBM_QUERYRANGE 20-5
SBM_SETHILITE 20-5
SBM_SETPOS 20-6
SBM_SETSCROLLBAR 20-7
SBM_SETTHUMBSIZE 20-7
SBS_* values 20-1
SB_* values 20-2, 20-3, 23-2, 23-5
Scale Matrix 4-231
SCP_* values 4-263
scroll bar control data 20-1
scroll bar control window processing 20-1
scroll bar styles 20-1
Scroll Window 9-263
SC_* values 15-18
SDW_* values 4-210, 4-327
SEGEM_* values 4-166, 4-283
segment attributes
chained 4-322
detectability 4-322
highlight 4-322
nonstore 4-322
store 4-322
transformability 4-322
visibility 4-322
segments
add comment 4-38
call matrix 4-25
close current 4-35
correlate 4-46
correlate chain 4-41
correlate section of chain 4-44
delete all 4-66
delete retained 4-65
draw 4-74
draw chain 4-70
draw section of chain 4-72
get graphic data from 4-87
open 4-104
query attributes 4-205
query initial attributes 4-175
query names 4-206

segments *(continued)*
query priority 4-207
query transform matrix 4-208
return last error during drawing 4-82
set attributes 4-321
set initial attributes 4-287
set priority 4-323
set transform matrix 4-325
SEGOFF 2-31
Send Message 9-266
Send Message Hook 10-18
Send Message to Dialog Item 9-265
SendMsgHook 10-18
SEPARATOR menu item 26-14
session title
query 9-217
Set Accelerator Table 9-267
Set Active Window 9-268
Set Arc Parameter 27-21
Set Arc Parameters 4-233
Set Attribute Mode 4-235
Set Attributes 4-237
Set Background Color 4-244, 27-22
Set Background Indexed Color 27-22
Set Background Mix 4-246, 27-23
Set Bit Map 4-248
Set Bit-Map Bits 4-249
Set Bit-Map Dimension 4-250
Set Bit-Map Identifier 4-251
Set Bits 9-269
Set Bits Under Mask 9-270
Set Capture 9-271
Set Character Angle 4-252, 27-24
Set Character Box 4-254
Set Character Cell 27-25
Set Character Direction 4-256, 27-26
Set Character Mode 4-258
Set Character Precision 27-26
Set Character Set 4-260, 27-27
Set Character Shear 4-261, 27-27
Set Class Message Interest 9-272
Set Clip Path 4-263, 27-28
Set Clip Region 4-265
Set Clipboard Data 9-273
Set Clipboard Owner 9-275
Set Clipboard Viewer 9-276
Set Code Page 4-269, 9-277
Set Color 4-267, 27-29
Set Current Position 4-270, 27-29
Set Default Arc Parameters 4-271
Set Default Attributes 4-272
Set Default Tag 4-277
Set Default View Matrix 4-275
Set Default Viewing Limits 4-278
Set Dialog Item Short 9-278
Set Dialog Item Text 9-279
Set Draw Control 4-279
Set Drawing Mode 4-281
Set Edit Mode 4-283
Set Element Pointer 4-284
Set Element Pointer At Label 4-285
Set Error Information 9-280
Set Extended Color 27-30
Set Focus 9-281
Set Fractional Line Width 27-30
Set Graphics Field 4-286
Set Hook 9-282
set identifier
delete 4-67
Set Indexed Color 27-31
Set Individual Attribute 27-32
Set Initial Segment Attributes 4-287
Set Keyboard State Table 9-283
Set Line End 4-289, 27-33
Set Line Join 4-291, 27-33
Set Line Type 4-293, 27-34
Set Line Width 4-295, 27-35
Set Line Width Geom 4-296
Set Marker 4-298
Set Marker Box 4-300
Set Marker Cell 27-35
Set Marker Precision 27-36
Set Marker Set 4-302, 27-36
Set Marker Symbol 27-37
Set Message Interest 9-284
Set Message Mode 9-285
Set Metafile Bits 4-303
Set Mix 4-304, 27-38
Set Model Transform 27-39
Set Model Transform Matrix 4-306
Set Multiple Window Positions 9-286
Set Owner 9-287
Set Page Viewport 4-308
Set Parent 9-288
Set Pattern 4-309
Set Pattern Reference Point 4-311, 27-40
Set Pattern Set 4-313, 27-40
Set Pattern Symbol 27-41
Set Pel 4-315
Set Pick Identifier 27-42
Set Pick-Aperture Position 4-316
Set Pick-Aperture Size 4-317
Set Pointer 9-289
Set Pointer Position 9-290
Set Presentation Parameter 9-291
Set Presentation Space 4-318
Set Rectangle 9-292
Set Rectangle Empty 9-293
Set Region 4-320
Set Segment Attributes 4-321
Set Segment Boundary 27-42
Set Segment Characteristics 27-43
Set Segment Priority 4-323
Set Segment Transform Matrix 4-325
Set Stop Draw 4-327
Set Stroke Line Width 27-44
Set Synchronization Mode 9-294
Set System Colors 9-295
Set System Modal Window 9-298
Set System Value 9-299
Set Tag 4-328
Set Value 9-301
Set Viewing Limits 4-329
Set Viewing Transform 27-44
Set Viewing Transform Matrix 4-331
Set Viewing Window 27-45
Set Window Enabled State 9-88
Set Window Position 9-303
Set Window Text 9-307
Set Window Word Bits 9-302
Set Window Word Long 9-308
Set Window Word Short 9-309
Set Window Words Pointer 9-306
SFACTORS 2-31

SGH_* values 9-11, 9-212
Sharp Fillet at Current Position 27-46
Sharp Fillet at Given Position 27-46
SHE_* values 2-29, 9-39
SHORT 2-31
Show Cursor 9-310
Show Pointer 9-311
Show Tracking Rectangle 9-312
Show Window 9-313
single-byte character sets 28-18
SIZEF 2-31
SIZEROF 2-31
SIZEROL 2-31
SMHSTRUCT 2-31
SMIM_* values 9-272, 9-284
SMI_* values 9-272, 9-284
SM_QUERYHANDLE 21-3
SM_SETHANDLE 21-3
Sound Alarm 9-13
source resource file 26-25
SplQmAbort 7-2
SplQmClose 7-3
SplQmEndDoc 7-4
SplQmOpen 7-5
SplQmStartDoc 7-6
SplQmWrite 7-7
SplQpInstall 7-8
SplQpQueryDt 7-9
SPL_* values 7-4, 7-5
spooler
 install queue processor 7-8
 query processor data types 7-9
 queue manager abort 7-2
 queue manager close 7-3
 queue manager end document 7-4
 queue manager open 7-5
 queue manager start document 7-6
 queue manager write 7-7
Spooler Queue Manager Abort 7-2
Spooler Queue Manager Close 7-3
Spooler Queue Manager End Document 7-4
Spooler Queue Manager Open 7-5
Spooler Queue Manager Start Document 7-6
Spooler Queue Manager Write 7-7
Spooler Queue Processor Install 7-8
Spooler Queue Processor Query Data Type 7-9
SPTR_* values 9-223
SS_* values 21-1
standard bit-map formats B-1
Start Installed Application 9-122
Start Timer 9-314
static control data 21-2
static control styles 21-1
static control window processing 21-1
Stop Timer 9-315
STORAGE 2-31
store attribute for segments
 modify (GpiSetSegmentAttrs) 4-322
STR 2-31
STRCOND 2-31
string
 convert to uppercase 9-329
strings
 load from resource 9-143
 substitute 9-317
STRINGTABLE statement 26-7
STRL 2-32
STRLLIST 2-32
Stroke Path 4-333
structures 2-1
 metalanguage 2-1
STR16 2-32
STR32 2-32
STR64 2-32
STR8 2-32
Subclass Window 9-316
submenus 26-13
Substitute Strings 9-317
Subtract Rectangle 9-318
suppression options 4-111
SUP_* values 4-111
SV_* values 9-224
SWBLOCK 2-32
SWCNTRL 2-32
SWENTRY 2-32
switch list
 remove entry 9-262
Switch To Program 9-319
SWL_* values 2-32
SWP 2-33
SWP_* values 2-33, 9-239, 9-303, 12-49
SW_* options 9-263
SYSCLR_* indexes 9-295
SYSINF_* values 9-234
system color
 query 9-221
 set 9-295
system pointer
 query 9-223
system value
 query 9-224
 set 9-299

T

tag
 query 4-211
 query default 4-160
 set 4-328
task title
 query 9-229
TBM_QUERYHILITE 22-3
TBM_SETHILITE 22-3
TBM_TRACKMOVE 22-4
templates
 dialog 26-18
 format 26-14
 menus 26-14
Terminate 9-320
Terminate Application 9-321
text
 draw 9-84
 query box 4-212
TF_* values 2-34
TID 2-33
TIME 2-33
timer
 start 9-314
title bar
 control data 22-1
 control window processing 22-1
 style 22-1
TRACKINFO 2-33

tracking rectangle
hide 9-312
show 9-312
transform matrix
query model 4-189
rotate 4-226
scale 4-231
set model 4-306
translate 4-335
transformability attribute for segments
modify (GpiSetSegmentAttrs) 4-322
transforms
set viewing 4-331
TRANSFORM_* values 4-25, 4-226, 4-231, 4-275, 4-306, 4-325, 4-331, 4-335
Translate Accelerator 9-326
Translate Character with Code Page 9-29
Translate Matrix 4-335
Translate String with Code Page 9-30
triplets D-2
TXTBOX_* values 4-213

U

UCHAR 2-34
ULONG 2-35
Union Rectangle 9-327
Unload Fonts 4-337
Unrealize Color Table 4-338
Up cursor key 9-324
update region
exclude 9-95
query 9-231
Update Window 9-328
Uppercase Character 9-330
Uppercase String 9-329
user-defined resources 26-3
USERBUTTON 2-35
USHORT 2-35

V

Validate Rectangle 9-331
Validate Region 9-332
VGA 3-17
view matrix
query default 4-158
viewing limits
query 4-214
query default 4-161
set 4-329
viewing transform
set default 4-275
viewing transforms
query 4-215
viewports
query page 4-192
vio
associate 8-2
create logical font 8-3
create presentation space 8-4
delete logical font 8-6
destroy the presentation space 8-7
get device cell size 8-8
origin 8-9
query fonts 8-10
query set identifiers 8-12
vio *(continued)*
set device cell size 8-13
set origin 8-14
show presentation space 8-15
Vio Associate 8-2
Vio Create Logical Font 8-3
Vio Create Presentation Space 8-4
Vio Delete Set Id 8-6
Vio Destroy Presentation Space 8-7
Vio Get Device Cell Size 8-8
Vio Get Origin 8-9
Vio Query Fonts 8-10
Vio Query Set Identifiers 8-12
Vio Set Device Cell Size 8-13
Vio Set Origin 8-14
Vio Show Presentation Space 8-15
VioAssociate 8-2
VioCreateLogFont 8-3
VioCreatePS 8-4
VioDeleteSetId 8-6
VioDestroyPS 8-7
VIOFONTCELLSIZE 2-35
VioGetDeviceCellSize 8-8
VioGetOrg 8-9
VioQueryFonts 8-10
VioQuerySetIds 8-12
VioSetDeviceCellSize 8-13
VioSetOrg 8-14
VioShowPS 8-15
VIOSIZECOUNT 2-35
virtual key definitions F-1
visibility attribute for segments
modify (GpiSetSegmentAttrs) 4-322
VK_* values 2-1, 9-109
VOID 2-35

W

Wait Message 9-333
WA_* values 9-13
WCS_* values 9-26
WC_* values 9-245, 11-1, 22-1
WIDTH4 2-35
WinAddAtom 9-10
WinAddProgram 9-11
WinAddSwitchEntry 9-12
WinAlarm 9-13
WinAllocMem 9-14
WinAssociateHelpInstance 9-15
WinAvailMem 9-16
WinBeginEnumWindows 9-17
WinBeginPaint 9-18
WinBroadcastMsg 9-19
WinCalcFrameRect 9-20
WinCallMsgFilter 9-21
WinCancelShutdown 9-22
WinCatch 9-23
WinChangeSwitchEntry 9-24
WinCloseClipbrd 9-25
WinCompareStrings 9-26
WinCopyAccelTable 9-27
WinCopyRect 9-28
WinCpTranslateChar 9-29
WinCpTranslateString 9-30
WinCreateAccelTable 9-31
WinCreateAtomTable 9-32
WinCreateCursor 9-33

WinCreateDataStructure 9-35
WinCreateDlg 9-37
WinCreateFrameControls 9-38
WinCreateGroup 9-39
WinCreateHeap 9-41
WinCreateHelpInstance 9-43
WinCreateHelpTable 9-44
WinCreateMenu 9-45
WinCreateMsgQueue 9-46
WinCreatePointer 9-47
WinCreatePointerIndirect 9-48
WinCreateStdWindow 9-49
WinCreateSwitchEntry 9-51
WinCreateWindow 9-52
WinDdeInitiate 9-55
WinDdePostMsg 9-56
WinDdeRespond 9-58
WinDefAVioWindowProc 9-59
WinDefDlgProc 9-60
WinDefWindowProc 9-61
WinDeleteAtom 9-62
WinDeleteLibrary 9-63
WinDeleteProcedure 9-64
WinDestroyAccelTable 9-65
WinDestroyAtomTable 9-66
WinDestroyCursor 9-67
WinDestroyDataStructure 9-68
WinDestroyHeap 9-69
WinDestroyHelpInstance 9-70
WinDestroyMsgQueue 9-71
WinDestroyPointer 9-72
WinDestroyWindow 9-73
WinDismissDlg 9-75
WinDispatchMsg 9-76
WinDlgBox 9-77
window
 create 9-52
 destroy 9-73
 query 9-235
 query active 9-171
 query class name 9-180
 query desktop 9-191
 query device context for 9-237
 query handle from device context 9-334
 query pointer 9-241
 query position 9-239
 query size 9-239
 query text 9-243
 query text length 9-244
 query unsigned long integer value of 9-245
 query unsigned short integer value of 9-246
 register class of 9-249
 scroll 9-263
 set message interest 9-284
 set multiple positions 9-286
 set owner 9-287
 set position 9-303
 set to system modal 9-298
 update 9-328
window class
 set message interest 9-272
window class styles 12-1
Window From Point 9-336
Window Procedure 10-3
window processing
 button control 13-1
 combo box control 19-1
window processing *(continued)*
 control 11-1
 default 11-1, 12-1
 entry field control 14-1
 frame control 15-1
 language support 12-54
 list box control 16-1
 menu control 17-1
 multi-line entry field control 18-1
 prompted entry field control 19-1
 scroll bar control 20-1
 static control 21-1
windows
 create standard 9-49
 create standard frame controls 9-38
 define procedure 10-3
 enable update 9-89
 find descendant 9-336
 get minimum position 9-111
 get multiples from identities 9-159
 invoke default procedure 9-61
 is handle valid 9-131
 lock 9-146
 map points 9-151
 open device context 9-163
 process message box 9-152
 query class information 9-179
 query descendancy 9-128
 query enabled state 9-132
 query handle from identifier 9-335
 query is child 9-128
 query lock count 9-238
 query object 9-199
 query rectangle 9-242
 query system modal 9-222
 query visibility 9-134
 set active 9-268
 set enabled state 9-88
 set parent 9-288
 set text 9-307
 set visibility state 9-89, 9-313
 show 9-313
 start flashing 9-98
 stop flashing 9-98
 unlock 9-146
WINDOWTEMPLATE statement 26-15
WinDrawBitmap 9-79
WinDrawBorder 9-81
WinDrawPointer 9-83
WinDrawText 9-84
WinEmptyClipbrd 9-86
WinEnablePhysInput 9-87
WinEnableWindow 9-88
WinEnableWindowUpdate 9-89
WinEndEnumWindows 9-90
WinEndPaint 9-91
WinEnumClipbrdFmts 9-92
WinEnumDlgItem 9-93
WinEqualRect 9-94
WinExcludeUpdateRegion 9-95
WinFillRect 9-96
WinFindAtom 9-97
WinFlashWindow 9-98
WinFocusChange 9-99
WinFreeErrorInfo 9-101
WinFreeMem 9-102
WinFreeMsg 9-103

WinGetClipPS 9-105
WinGetCurrentTime 9-106
WinGetDlgMsg 9-107
WinGetErrorInfo 9-108
WinGetKeyState 9-109
WinGetLastError 9-110
WinGetMinPosition 9-111
WinGetMsg 9-112
WinGetNextWindow 9-114
WinGetPhysKeyState 9-115
WinGetPS 9-116
WinGetScreenPS 9-117
WinGetSysBitmap 9-118
WinInflateRect 9-119
WinInitialize 9-120
WinInSendMsg 9-121
WinInstStartApp 9-122
WinIntersectRect 9-124
WinInvalidateRect 9-125
WinInvalidateRegion 9-126
WinInvertRect 9-127
WinIsChild 9-128
WinIsRectEmpty 9-129
WinIsThreadActive 9-130
WinIsWindow 9-131
WinIsWindowEnabled 9-132
WinIsWindowShowing 9-133
WinIsWindowVisible 9-134
WinLoadAccelTable 9-135
WinLoadDlg 9-136
WinLoadHelpTable 9-138
WinLoadLibrary 9-139
WinLoadMenu 9-140
WinLoadPointer 9-141
WinLoadProcedure 9-142
WinLoadString 9-143
WinLockHeap 9-144
WinLockVisRegions 9-145
WinLockWindow 9-146
WinLockWindowUpdate 9-147
WinMakePoints 9-148
WinMakeRect 9-149
WinMapDlgPoints 9-150
WinMapWindowPoints 9-151
WinMessageBox 9-152
WinModifyDataStructure 9-155
WinMsgMuxSemWait 9-157
WinMsgSemWait 9-158
WinMultWindowFromIDs 9-159
WinNextChar 9-160
WinOffsetRect 9-161
WinOpenClipbrd 9-162
WinOpenWindowDC 9-163
WinPeekMsg 9-164
WinPostMsg 9-165
WinPostQueueMsg 9-166
WinPrevChar 9-167
WinProcessDlg 9-168
WinPtInRect 9-169
WinQueryAccelTable 9-170
WinQueryActiveWindow 9-171
WinQueryAnchorBlock 9-172
WinQueryAtomLength 9-173
WinQueryAtomName 9-174
WinQueryAtomUsage 9-175
WinQueryBits 9-176
WinQueryBitsUnderMask 9-177
WinQueryCapture 9-178
WinQueryClassInfo 9-179
WinQueryClassName 9-180
WinQueryClipbrdData 9-181
WinQueryClipbrdFmtInfo 9-182
WinQueryClipbrdOwner 9-183
WinQueryClipbrdViewer 9-184
WinQueryCp 9-185
WinQueryCpList 9-186
WinQueryCursorInfo 9-187
WinQueryDataStructure 9-188
WinQueryDefinition 9-190
WinQueryDesktopWindow 9-191
WinQueryDlgItemShort 9-192
WinQueryDlgItemText 9-193
WinQueryDlgItemTextLength 9-194
WinQueryFocus 9-195
WinQueryHelpInstance 9-196
WinQueryMsgPos 9-197
WinQueryMsgTime 9-198
WinQueryObjectWindow 9-199
WinQueryPointer 9-200
WinQueryPointerInfo 9-201
WinQueryPointerPos 9-202
WinQueryPresParam 9-203
WinQueryProfileData 9-205
WinQueryProfileInt 9-207
WinQueryProfileSize 9-208
WinQueryProfileString 9-210
WinQueryProgramTitles 9-212
WinQueryQueueInfo 9-214
WinQueryQueueStatus 9-215
WinQuerySessionTitle 9-217
WinQuerySwitchEntry 9-218
WinQuerySwitchHandle 9-219
WinQuerySwitchList 9-220
WinQuerySysColor 9-221
WinQuerySysModalWindow 9-222
WinQuerySysPointer 9-223
WinQuerySystemAtomTable 9-227
WinQuerySysValue 9-224
WinQueryTaskSizePos 9-228
WinQueryTaskTitle 9-229
WinQueryUpdateRect 9-230
WinQueryUpdateRegion 9-231
WinQueryValue 9-232
WinQueryVersion 9-234
WinQueryWindow 9-235
WinQueryWindowDC 9-237
WinQueryWindowLockCount 9-238
WinQueryWindowPos 9-239
WinQueryWindowProcess 9-240
WinQueryWindowPtr 9-241
WinQueryWindowRect 9-242
WinQueryWindowText 9-243
WinQueryWindowTextLength 9-244
WinQueryWindowULong 9-245
WinQueryWindowUShort 9-246
WinReallocMem 9-248
WinRegisterClass 9-249
WinRegisterUserDatatype 9-251
WinRegisterUserMsg 9-256
WinRegisterWindowDestroy 9-258
WinReleaseHook 9-259
WinReleasePS 9-260
WinRemovePresParam 9-261
WinRemoveSwitchEntry 9-262

WinScrollWindow 9-263
WinSendDlgItemMsg 9-265
WinSendMsg 9-266
WinSetAccelTable 9-267
WinSetActiveWindow 9-268
WinSetBits 9-269
WinSetBitsUnderMask 9-270
WinSetCapture 9-271
WinSetClassMsgInterest 9-272
WinSetClipbrdData 9-273
WinSetClipbrdOwner 9-275
WinSetClipbrdViewer 9-276
WinSetCp 9-277
WinSetDlgItemShort 9-278
WinSetDlgItemText 9-279
WinSetErrorInfo 9-280
WinSetFocus 9-281
WinSetHook 9-282
WinSetKeyboardStateTable 9-283
WinSetMsgInterest 9-284
WinSetMsgMode 9-285
WinSetMultWindowPos 9-286
WinSetOwner 9-287
WinSetParent 9-288
WinSetPointer 9-289
WinSetPointerPos 9-290
WinSetPresParam 9-291
WinSetRect 9-292
WinSetRectEmpty 9-293
WinSetSynchroMode 9-294
WinSetSysColors 9-295
WinSetSysModalWindow 9-298
WinSetSysValue 9-299
WinSetValue 9-301
WinSetWindowBits 9-302
WinSetWindowPos 9-303
WinSetWindowPtr 9-306
WinSetWindowText 9-307
WinSetWindowULong 9-308
WinSetWindowUShort 9-309
WinShowCursor 9-310
WinShowPointer 9-311
WinShowTrackRect 9-312
WinShowWindow 9-313
WinStartTimer 9-314
WinStopTimer 9-315
WinSubclassWindow 9-316
WinSubstituteStrings 9-317
WinSubtractRect 9-318
WinSwitchToProgram 9-319
WinTerminate 9-320
WinTerminateApp 9-321
WinThrow 9-322
WinTrackRect 9-323
WinTranslateAccel 9-326
WinUnionRect 9-327
WinUpdateWindow 9-328
WinUpper 9-329
WinUpperChar 9-330
WinValidateRect 9-331
WinValidateRegion 9-332
WinWaitMsg 9-333
WinWindowFromDC 9-334
WinWindowFromID 9-335
WinWindowFromPoint 9-336
WinWriteProfileData 9-337
WinWriteProfileString 9-339

WM_ACTIVATE 9-73, 9-305, 12-3
 frame control window processing 15-6
 language support dialog processing 12-56
 language support window processing 12-54
WM_ADJUSTWINDOWPOS 9-305, 12-4
WM_APPTERMINATENOTIFY 12-5
WM_BUTTON1DBLCLK 12-6
 frame control window processing 15-6
 multi-line entry field window processing 18-35
WM_BUTTON1DOWN 12-7
 frame control window processing 15-6
 multi-line entry field window processing 18-36
WM_BUTTON1UP 12-7
 frame control window processing 15-7
 multi-line entry field window processing 18-36
WM_BUTTON2DBLCLK 12-8
 frame control window processing 15-6
WM_BUTTON2DOWN 12-9
 frame control window processing 15-7
WM_BUTTON2UP 12-9
 frame control window processing 15-7
WM_BUTTON3DBLCLK 12-10
WM_BUTTON3DOWN 12-11
WM_BUTTON3UP 12-11
WM_CALCFRAMERECT 12-12
 frame control window processing 15-8
WM_CALCVALIDRECTS 12-12
WM_CHAR 12-14
 default dialog processing 12-50
 entry field control window processing 14-10
 frame control window processing 15-8
 list box control window processing 16-16
 multi-line entry field window processing 18-37
WM_CLOSE 12-16
 default dialog processing 12-50
 frame control window processing 15-8
WM_COMMAND 11-2, 12-16
 button control window processing 13-2
 default dialog processing 12-51
 menu control window processing 17-3
 title bar control window processing 22-2
WM_CONTROL 11-2, 12-17
 button control window processing 13-3
 entry field control window processing 14-3
 language support dialog processing 12-56
 language support window processing 12-54
 list box control window processing 16-2
 multi-line entry field window processing 18-4
 prompted entry field window processing 19-2
WM_CONTROLHEAP 12-17
WM_CONTROLPOINTER 12-18
WM_CREATE 12-18
WM_DDE_ACK 24-1
WM_DDE_ADVISE 24-2
WM_DDE_DATA 24-3
WM_DDE_EXECUTE 24-3
WM_DDE_INITIATE 24-4
WM_DDE_INITIATEACK 24-4
WM_DDE_POKE 24-5
WM_DDE_REQUEST 24-6
WM_DDE_TERMINATE 24-6
WM_DDE_UNADVISE 24-7
WM_DESTROY 9-73, 12-19
WM_DESTROYCLIPBOARD 23-1
WM_DRAWCLIPBOARD 23-1
WM_DRAWITEM 12-20

WM_DRAWITEM *(continued)*
frame control window processing 15-9
list box control window processing 16-3
menu control window processing 17-3
WM_ENABLE 12-20
button control window processing 13-9
multi-line entry field window processing 18-39
WM_ERASEBACKGROUND
frame control window processing 15-9
WM_ERROR 12-21
WM_FLASHWINDOW
frame control window processing 15-10
WM_FOCUSCHANGE 12-21
frame control window processing 15-10
WM_FORMATFRAME 12-22
frame control window processing 15-11
WM_HELP 11-2, 12-22
button control window processing 13-4
menu control window processing 17-4
WM_HITTEST 12-23
WM_HSCROLL 12-24
scroll bar window processing 20-2
WM_HSCROLLCLIPBOARD 23-2
WM_INITDLG 12-24
default dialog processing 12-51
WM_INITMENU 12-24
frame control window processing 15-12
menu control window processing 17-4
WM_JOURNALNOTIFY 12-24
WM_MATCHMNEMONIC 12-25
button control window processing 13-10
default dialog processing 12-52
static control window processing 21-4
WM_MEASUREITEM 12-25
frame control window processing 15-12
list box control window processing 16-4
menu control window processing 17-5
WM_MENUEND 12-26
menu control window processing 17-5
WM_MENUSELECT 12-26
frame control window processing 15-12
menu control window processing 17-6
WM_MINMAXFRAME 12-26
frame control window processing 15-3
WM_MOUSEMOVE 12-26
multi-line entry field window processing 18-39
WM_MOVE 9-305, 12-27
WM_NEXTMENU 12-28
frame control window processing 15-12
menu control window processing 17-7
WM_NULL 12-28
WM_OTHERWINDOWDESTROYED 9-73, 12-28
WM_PACTIVATE 12-29
WM_PAINT 12-30
frame control window processing 15-13
language support dialog processing 12-56
language support window processing 12-54
WM_PAINTCLIPBOARD 23-3
WM_PCONTROL 12-29
WM_PPAINT 12-30
language support dialog processing 12-57
language support window processing 12-55
WM_PRESPARAMCHANGED 12-31
WM_PSETFOCUS 12-31
WM_PSIZE 12-32
WM_PSYSCOLORCHANGE 12-32
WM_QUERYACCELTABLE 12-33
WM_QUERYBORDERSIZE
frame control window processing 15-13
WM_QUERYCONVERTPOS 12-33
button control window processing 13-10
control window processing 22-5
entry field control window processing 14-11
frame control window processing 15-14
list box control window processing 16-17
menu control window processing 17-21
scroll bar window processing 20-8
static control window processing 21-4
WM_QUERYDLGCODE
default dialog processing 12-52
WM_QUERYFOCUSCHAIN
frame control window processing 15-14
WM_QUERYFRAMECTLCOUNT
frame control window processing 15-15
WM_QUERYFRAMEINFO
frame control window processing 15-15
WM_QUERYICON
frame control window processing 15-16
WM_QUERYTRACKINFO 12-34
title bar control window processing 22-2
WM_QUERYWINDOWPARAMS 12-35
button control window processing 13-10
entry field control window processing 14-11
frame control window processing 15-16
list box control window processing 16-17
menu control window processing 17-21
multi-line entry field window processing 18-40
scroll bar window processing 20-8
static control window processing 21-4
title bar control window processing 22-5
WM_QUIT 12-35
WM_RENDERALLFMTS 9-73, 23-3
WM_RENDERFMT 23-4
WM_SAVEAPPLICATION 12-36
WM_SEM1 12-37
WM_SEM2 12-37
WM_SEM3 12-38
WM_SEM4 12-38
WM_SETACCELTABLE 12-39
WM_SETBORDERSIZE
frame control window processing 15-17
WM_SETFOCUS 12-39
language support dialog processing 12-57
language support window processing 12-55
WM_SETICON
frame control window processing 15-17
WM_SETSELECTION 12-40
WM_SETWINDOWPARAMS 12-40
button control window processing 13-11
entry field control window processing 14-11
frame control window processing 15-18
list box control window processing 16-17
menu control window processing 17-22
multi-line entry field window processing 18-40
scroll bar window processing 20-9
static control window processing 21-5
title bar control window processing 22-5
WM_SHOW 12-41
WM_SIZE 9-305, 12-42
frame control window processing 15-18
language support dialog processing 12-57
language support window processing 12-55

WM_SIZECLIPBOARD 23-4
WM_SUBSTITUTESTRING 12-42
WM_SYSCOLORCHANGE 12-43
 language support dialog processing 12-58
 language support window processing 12-55
WM_SYSCOMMAND 12-43
 button control window processing 13-4
 frame control window processing 15-18
 menu control window processing 17-7
WM_SYSVALUECHANGED 12-44
WM_TIMER 12-46
WM_TRACKFRAME 12-46
 frame control window processing 15-19
WM_TRANSLATEACCEL 12-47
 frame control window processing 15-20
WM_TRANSLATEMNEMONIC 12-47
 frame control window processing 15-20
WM_UPDATEFRAME 12-48
 frame control window processing 15-20
WM_UPDATESTYLE
 prompted entry field window processing 19-6
WM_VSCROLL 12-48
 scroll bar window processing 20-3
WM_VSCROLLCLIPBOARD 23-5
WM_WINDOWPOSCHANGED 12-48
WM_* messages 9-215
WNDPARAMS 2-35
WNDPROC 2-35, 10-3
World Coordinates Bit Blt 4-339
WPM_* values 2-35
WPOINT 2-35
WRECT 2-36
Write Profile Data 6-25, 9-337
Write Profile String 6-26, 9-339
WS_* values 9-116, 12-2

X

XYF_* values 2-36
XYWINSIZE 2-36

IBM United Kingdom
International Products Limited
PO Box 41, North Harbour
Portsmouth, PO6 3AU
England

Printed in Denmark by
visoprint as

